AWS SDK for PHP

version 2.5.2

Amazon Web Services

January 29, 2014

Contents

AWS SDK for PHP **1**

- Signing Up for AWS 1
  - Creating an AWS account 1
    - To sign up for AWS 1
    - To view your AWS credentials 1
  - Getting your AWS credentials 1
- Requirements 2
  - Minimum requirements 2
  - Optimal settings 2
  - Compatibility test 2
- Installation 2
  - Installing via Composer 2
  - Installing via Phar 3
  - Installing via Zip 4
    - Installing via PEAR 4
- Getting Started Guide 4
  - Including the SDK 4
  - Creating a client object 5
    - Factory method 5
    - Service builder 6
  - Performing service operations 6
  - Working with modeled responses 7
  - Detecting and handling errors 7
  - Waiters 7
  - Iterators 8
- Migration Guide 8
  - Introduction 8
  - Which Services are Supported? 8
  - What's New? 8
  - What's Different? 9
    - Architecture 9
    - Project Dependencies 9
    - Namespaces 9
    - Coding Standards 9
    - Required Regions 10
    - Client Factories 10
    - Configuration 10
    - Service Operations 10
    - Responses 11
    - Exceptions 11

Iterators 12
Comparing Code Samples from Both SDKs 12
Example 1 - Amazon S3 ListParts Operation 12
From Version 1 of the SDK 12
From Version 2 of the SDK 13
Example 2 - Amazon DynamoDB Scan Operation 13
From Version 1 of the SDK 13
From Version 2 of the SDK 15
Side-by-side Guide 15
Installing and Including the SDKs 15
Using Composer 15
Without Composer 16
Configuring and Instantiating the SDKs 16
Instantiating Clients via the Service Builder 16
Instantiating Clients via Client Factories 17
Complete Examples 17
Example 1 - Dual Amazon S3 Clients 18
Example 2 - Amazon DynamoDB and Amazon SNS Clients 18
Final Notes 19
Providing Credentials to the SDK 19
Introduction 19
Which technique should you choose? 19
Using credentials from environment variables 20
Using IAM roles for Amazon EC2 instances 20
Caching IAM role credentials 20
Setting credentials explicitly in your code 21
Using a configuration file with the service builder 21
Passing credentials into a client factory method 22
Setting credentials after instantiation 23
Using temporary credentials from AWS STS 23
Getting temporary credentials 24
Providing temporary credentials to the SDK 24
Configuring the SDK 25
Configuration files 25
How configuration files work 25
Using a custom configuration file 26
Client configuration options 27
Specifying a region 28
Setting a custom endpoint 28
Using a proxy 29
Command Objects 29
Typical SDK usage 29

A peek under the hood 29
Using command objects 30
Manipulating command objects before execution 30
Request and response objects 30
Managing command state 30
Using requests and responses 31
Executing commands in parallel 31
Error handling with parallel commands 32
Waiters 32
Introduction 32
Basic Configuration 32
Waiter Objects 33
Waiter Events 33
Custom Waiters 33
Waiter Definitions 34
Iterators 35
Introduction 35
Iterator Objects 36
Basic Configuration 36
Iterator Events 36
Modeled Responses 37
Introduction 37
Working with Model objects 37
Getting nested values 38
Using data in the model 38
Getting Response Headers 39
Static Client Facades 39
Introduction 39
Why Use Client Facades? 40
Enabling and Using Client Facades 40
Testing Code that Uses Client Facades 41
Performance Guide 42
Upgrade PHP 42
Use PHP 5.5 or an opcode cache like APC 42
APC 43
Installing on Amazon Linux 43
Modifying APC settings 43
apc.shm_size=128M 43
apc.stat=0 43
Use Composer with a classmap autoloader 44
Uninstall Xdebug 44
Install PECL uri_template 44

Turn off parameter validation 44
Cache instance profile credentials 45
Check if you are being throttled 45
Preload frequently included files 45
Profile your code to find performance bottlenecks 46
Comparing SDK1 and SDK2 46
Comparing batch requests 46
Frequently Asked Questions (FAQ) 46
What methods are available on a client? 46
What do I do about a cURL SSL certificate error? 46
How do I disable SSL? 47
How can I make the SDK faster? 47
Why can't I upload or download files greater than 2GB? 47
How can I see what data is sent over the wire? 47
How can I set arbitrary headers on a request? 48
Does the SDK follow semantic versioning? 48
Why am I seeing a "Cannot redeclare class" error? 49
What is an InstanceProfileCredentialsException? 49
Auto Scaling 49
Creating a client 49
Factory method 49
Service builder 50
API Reference 50
AWS CloudFormation 51
Creating a client 51
Factory method 51
Service builder 51
API Reference 52
Amazon CloudFront 52
Creating a client 52
Factory method 52
Service builder 52
Signing CloudFront URLs for Private Distributions 53
API Reference 55
Amazon CloudFront (2012-05-05) 55
Creating a client 55
Factory method 55
Service builder 55
Signing CloudFront URLs for Private Distributions 56
API Reference 58
Amazon CloudSearch 58
Creating a client 58

Factory method 58
Service builder 59
API Reference 59
AWS CloudTrail 59
Creating a client 59
Factory method 59
Service builder 60
Blog articles 60
API Reference 60
Amazon CloudWatch 60
Creating a client 61
Factory method 61
Service builder 61
API Reference 61
AWS Data Pipeline 62
Creating a client 62
Factory method 62
Service builder 62
API Reference 63
AWS Direct Connect 63
Creating a client 63
Factory method 63
Service builder 63
API Reference 64
Amazon DynamoDB 64
Creating a client 64
Factory method 64
Service builder 65
Creating tables 65
Updating a table 66
Describing a table 66
Listing tables 66
Iterating over all tables 67
Adding items 67
Retrieving items 68
Query and scan 68
Query 68
Scan 69
Deleting items 70
Deleting a table 70
Local secondary indexes 70
Using the WriteRequestBatch 72

API Reference 73
Amazon DynamoDB (2011-12-05) 73
Creating a client 73
Factory method 73
Service builder 74
Creating tables 74
Updating a table 74
Describing a table 75
Listing tables 75
Iterating over all tables 75
Adding items 76
Retrieving items 76
Query and scan 77
Query 77
Scan 78
Deleting a table 78
Using the WriteRequestBatch 78
API Reference 79
Amazon Elastic Compute Cloud 79
Creating a client 79
Factory method 79
Service builder 80
API Reference 80
Amazon ElastiCache 82
Creating a client 82
Factory method 82
Service builder 83
API Reference 83
AWS Elastic Beanstalk 84
Creating a client 84
Factory method 84
Service builder 84
API Reference 85
Elastic Load Balancing 85
Creating a client 85
Factory method 85
Service builder 86
API Reference 86
Amazon Elastic Transcoder 86
Creating a client 87
Factory method 87
Service builder 87

API Reference 87
Amazon Elastic MapReduce 88
Creating a client 88
Factory method 88
Service builder 88
API Reference 89
Amazon Glacier 89
Creating a client 89
Factory method 89
Service builder 89
API Reference 90
AWS Identity and Access Management 90
Creating a client 90
Factory method 90
Service builder 91
API Reference 91
AWS Import/Export 92
Creating a client 92
Factory method 92
Service builder 93
API Reference 93
Amazon Kinesis 93
Creating a client 93
Factory method 93
Service builder 94
Creating a stream 94
API Reference 94
AWS OpsWorks 95
Creating a client 95
Factory method 95
Service builder 95
API Reference 95
Amazon Relational Database Service 96
Creating a client 96
Factory method 96
Service builder 97
API Reference 97
Amazon Redshift 98
Creating a client 98
Factory method 98
Service builder 98
Creating a cluster 99

Creating snapshots 99
Events 99
API Reference 100
Amazon Route 53 100
Creating a client 101
Factory method 101
Service builder 101
API Reference 101
Amazon Simple Storage Service 102
Creating a client 102
Factory method 102
Service builder 102
Creating a bucket 102
Creating a bucket in another region 102
Waiting until the bucket exists 103
Uploading objects 103
Uploading a file 103
Uploading from a stream 104
Listing your buckets 104
Listing objects in your buckets 105
Downloading objects 105
Saving objects to a file 106
Uploading large files using multipart uploads 106
Setting ACLs and Access Control Policies 107
Creating a pre-signed URL 107
Amazon S3 stream wrapper 109
Syncing data with Amazon S3 109
Uploading a directory to a bucket 109
Customizing the upload sync 109
More control with the UploadSyncBuilder 110
Downloading a bucket to a directory 110
Customizing the download sync 110
More control with the DownloadSyncBuilder 111
Cleaning up 111
API Reference 111
Amazon Simple Email Service 112
Creating a client 112
Factory method 112
Service builder 112
API Reference 113
Amazon SimpleDB 113
Creating a client 113

Factory method 113
Service builder 114
Creating domains 114
List all domains 114
Retrieving a domain 114
Adding items 114
Retrieving items 115
GetAttributes 115
Select 115
Deleting items 116
Deleting domains 116
API Reference 116
Amazon Simple Notification Service 116
Creating a client 116
Factory method 116
Service builder 117
API Reference 117
Amazon Simple Queue Service 118
Creating a client 118
Factory method 118
Service builder 118
Creating a queue 118
Sending messages 119
Receiving messages 119
API Reference 120
AWS Storage Gateway 120
Creating a client 120
Factory method 120
Service builder 121
API Reference 121
AWS Security Token Service 122
Creating a client 122
Factory method 122
Service builder 122
Getting Temporary Credentials 122
Using Temporary Credentials 123
API Reference 124
AWS Support 124
Creating a client 124
Factory method 124
Service builder 124
API Reference 125

Amazon Simple Workflow Service 125
Creating a client 125
Factory method 125
Service builder 126
API Reference 126
DynamoDB Session Handler 127
Introduction 127
Basic Usage 127
1. Register the handler 127
2. Create a table for storing your sessions 127
3. Use PHP sessions like normal 128
Configuration 128
Pricing 129
Session Locking 130
Garbage Collection 130
Best Practices 131
Amazon S3 Stream Wrapper 131
Introduction 131
Downloading data 131
Opening Seekable streams 132
Uploading data 132
fopen modes 132
Other object functions 133
Working with buckets 134
Listing the contents of a bucket 134
Getting Started 135
In-Depth Guides 135
Service-Specific Guides 136
Articles from the Blog 138
Presentations 139
Slides 139
Videos 139

AWS SDK for PHP

Signing Up for AWS

> ***Important***
>
> This page is obsolete. Please see About Access Keys.

Creating an AWS account

Before you begin, you need to create an account. When you sign up for AWS, AWS signs your account up for all services. You are charged only for the services you use.

To sign up for AWS

1. Go to http://aws.amazon.com and click **Sign Up Now**.
2. Follow the on-screen instructions.

AWS sends you a confirmation email after the sign-up process is complete. At any time, you can view your current account activity and manage your account at http://aws.amazon.com/account. From the **My Account** page, you can view current charges and account activity and download usage reports.

To view your AWS credentials

1. Go to http://aws.amazon.com/.
2. Click **My Account/Console**, and then click **Security Credentials**.
3. Under **Your Account**, click **Security Credentials**.
4. In the spaces provided, type your user name and password, and then click **Sign in using our secure server**.
5. Under **Access Credentials**, on the **Access Keys** tab, your access key ID is displayed. To view your secret key, under **Secret Access Key**, click **Show**.

Your secret key must remain a secret that is known only by you and AWS. Keep it confidential in order to protect your account. Store it securely in a safe place, and never email it. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your secret key.

Getting your AWS credentials

In order to use the AWS SDK for PHP, you need your AWS Access Key ID and Secret Access Key.

To get your AWS Access Key ID and Secret Access Key

- Go to http://aws.amazon.com/.
- Click **Account** and then click **Security Credentials**. The Security Credentials page displays (you might be prompted to log in).
- Scroll down to Access Credentials and make sure the **Access Keys** tab is selected. The AWS Access Key ID appears in the Access Key column.
- To view the Secret Access Key, click **Show**.

Note

Important: Your Secret Access Key is a secret, which only you and AWS should know. It is important to keep it confidential to protect your account. Store it securely in a safe place. Never include it in your requests to AWS, and never e-mail it to anyone. Do not share it outside your organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon will ever ask you for your Secret Access Key.

Requirements

Aside from a baseline understanding of object-oriented programming in PHP (including PHP 5.3 namespaces), there are a few minimum system requirements to start using the AWS SDK for PHP. The extensions listed are common and are installed with PHP 5.3 by default in most environments.

Minimum requirements

- PHP 5.3.3+ compiled with the cURL extension
- A recent version of cURL 7.16.2+ compiled with OpenSSL and zlib

Note

To work with Amazon CloudFront private distributions, you must have the OpenSSL PHP extension to sign private CloudFront URLs.

Optimal settings

Please consult the *Performance Guide* for a list of recommendations and optimal settings that can be made to ensure that you are using the SDK as efficiently as possible.

Compatibility test

Run the *compatibility-test.php* file in the SDK to quickly check if your system is capable of running the SDK. In addition to meeting the minimum system requirements of the SDK, the compatibility test checks for optional settings and makes recommendations that can help you to improve the performance of the SDK. The compatibility test can output text for the command line or a web browser. When running in a browser, successful checks appear in green, warnings in purple, and failures in red. When running from the CLI, the result of a check will appear on each line.

When reporting an issue with the SDK, it is often helpful to share information about your system. Supplying the output of the compatibility test in forum posts or GitHub issues can help to streamline the process of identifying the root cause of an issue.

Installation

Installing via Composer

Using Composer is the recommended way to install the AWS SDK for PHP. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project needs and installs them into your project. In order to use the SDK with Composer, you must do the following:

1. Add `"aws/aws-sdk-php"` as a dependency in your project's `composer.json` file.

```
{
    "require": {
        "aws/aws-sdk-php": "2.*"
```

```
    }
}
```

Consider tightening your dependencies to a known version (e.g., `2.5.*`).

2. Download and install Composer.

```
curl -sS https://getcomposer.org/installer | php
```

3. Install your dependencies.

```
php composer.phar install
```

4. Require Composer's autoloader.

Composer prepares an autoload file that's capable of autoloading all of the classes in any of the libraries that it downloads. To use it, just add the following line to your code's bootstrap process.

```
require '/path/to/sdk/vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

During your development, you can keep up with the latest changes on the master branch by setting the version requirement for the SDK to `dev-master`.

```
{
   "require": {
      "aws/aws-sdk-php": "dev-master"
   }
}
```

If you are deploying your application to AWS Elastic Beanstalk, and you have a `composer.json` file in the root of your package, then Elastic Beanstalk will automatically perform a Composer `install` when you deploy your application.

Installing via Phar

Each release of the AWS SDK for PHP ships with a pre-packaged phar (PHP archive) file containing all of the classes and dependencies you need to run the SDK. Additionally, the phar file automatically registers a class autoloader for the AWS SDK for PHP and all of its dependencies when included. Bundled with the phar file are the following required and suggested libraries:

- Guzzle for HTTP requests
- Symfony2 EventDispatcher for events
- Monolog and Psr\Log for logging
- Doctrine for caching

You can download the packaged Phar and simply include it in your scripts to get started:

```
require '/path/to/aws.phar';
```

If you have phing installed, you can clone the SDK and build a phar file yourself using the *"phar"* task.

Note

If you are using PHP with the Suhosin patch (especially common on Ubuntu and Debian distributions), you may need to enable the use of phars in the `suhosin.ini`. Without this, including a phar file in your code will cause it to silently fail. You should modify the `suhosin.ini` file by adding the line:

```
suhosin.executor.include.whitelist = phar
```

Installing via Zip

Each release of the AWS SDK for PHP (since 2.3.2) ships with a zip file containing all of the classes and dependencies you need to run the SDK in a PSR-0 compatible directory structure. Additionally, the zip file includes a class autoloader for the AWS SDK for PHP and the following required and suggested libraries:

- Guzzle for HTTP requests
- Symfony2 EventDispatcher for events
- Monolog and Psr\Log for logging
- Doctrine for caching

Using the zip file is great if you:

1. Prefer not to or cannot use package managers like Composer and PEAR.
2. Cannot use phar files due to environment limitations.
3. Want to use only specific files from the SDK.

To get started, you must download the zip file, unzip it into your project to a location of your choosing, and include the autoloader:

```
require '/path/to/aws-autoloader.php';
```

Alternatively, you can write your own autoloader or use an existing one from your project.

If you have phing installed, you can clone the SDK and build a zip file yourself using the *"zip"* task.

Installing via PEAR

PEAR packages are easy to install, and are available in your PHP environment path so that they are accessible to any PHP project. PEAR packages are not specific to your project, but rather to the machine they're installed on.

From the command-line, you can install the SDK with PEAR as follows (this might need to be run as `sudo`):

```
pear config-set auto_discover 1
pear channel-discover pear.amazonwebservices.com
pear install aws/sdk
```

Alternatively, you can combine all three of the preceding statements into one by doing the following:

```
pear -D auto_discover=1 install pear.amazonwebservices.com/sdk
```

Once the SDK has been installed via PEAR, you can include the `aws.phar` into your project with:

```
require 'AWSSDKforPHP/aws.phar';
```

This assumes that the PEAR directory is in your PHP include path, which it probably is, if PEAR is working correctly. If needed, you can determine your PEAR directory by running `pear config-get php_dir`.

Getting Started Guide

This "Getting Started Guide" focuses on basic usage of the **AWS SDK for PHP**. After reading through this material, you should be familiar with the SDK and be able to start using the SDK in your application. This guide assumes that you have already *downloaded and installed the SDK* and retrieved your AWS access keys.

Including the SDK

No matter which technique you have used to to install the SDK, the SDK can be included into your project or script with just a single include (or require) statement. Please refer to the following table for the PHP code that best fits your installation technique. Please replace any instances of `/path/to/` with the actual path on your system.

Installation Technique	Include Statement

Using Composer	`require '/path/to/vendor/autoload.php';`
Using the Phar	`require '/path/to/aws.phar';`
Using the Zip	`require '/path/to/aws-autoloader.php';`
Using PEAR	`require 'AWSSDKforPHP/aws.phar';`

For the remainder of this guide, we will show examples that use the Composer installation method. If you are using a different installation method, then you can refer to this section and substitute in the proper code.

Creating a client object

To use the SDK, you first you need to instantiate a **client** object for the service you are using. We'll use the Amazon Simple Storage Service (Amazon S3) client as an example. You can instantiate a client using two different techniques.

Factory method

The easiest way to get up and running quickly is to use the web service client's `factory()` method and provide your AWS **credentials** (e.g., `key` and `secret`).

```
<?php

// Include the SDK using the Composer autoloader
require 'vendor/autoload.php';

use Aws\S3\S3Client;

// Instantiate the S3 client with your AWS credentials
$s3Client = S3Client::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
));
```

You can provide your credentials explicitly like in the preceding example, or you can choose to omit them if you are relying on **instance profile credentials** provided via AWS Identity and Access Management (AWS IAM) roles for EC2 instances, or **environment credentials** sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables. For more information about credentials, see *Providing Credentials to the SDK*.

> **Note**
>
> Instance profile credentials and other temporary credentials generated by the AWS Security Token Service (AWS STS) are not supported by every service. Please check if the service you are using supports temporary credentials by reading AWS Services that Support AWS STS.

Depending on the service, you may also need to provide a **region** value to the `factory()` method. The region value is used by the SDK to determine the regional endpoint to use to communicate with the service. Amazon S3 does not require you to provide a region, but other services like Amazon Elastic Compute Cloud (Amazon EC2) do. You can specify a region and other configuration settings along with your credentials in the array argument that you provide.

```
$ec2Client = \Aws\Ec2\Ec2Client::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
    'region' => 'us-east-1',
));
```

To know if the service client you are using requires a region and to find out which regions are supported by the client, please see the appropriate *service-specific guide*.

Service builder

Another way to instantiate a service client is using the `Aws\Common\Aws` object (a.k.a the **service builder**). The `Aws` object is essentially a service locator, and allows you to specify credentials and configuration settings such that they can be shared across all client instances. Also, every time you fetch a client object from the `Aws` object, it will be exactly the same instance.

```
use Aws\Common\Aws;

// Create a service locator using a configuration file
$aws = Aws::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
    'region' => 'us-east-1',
));

// Get client instances from the service locator by name
$s3Client = $aws->get('s3');
$ec2Client = $aws->get('ec2');

// The service locator always returns the same instance
$anotherS3Client = $aws->get('s3');
assert('$s3Client === $anotherS3Client');
```

You can also declare your credentials and settings in a **configuration file**, and provide the path to that file (in either php or json format) when you instantiate the `Aws` object.

```
// Create a `Aws` object using a configuration file
$aws = Aws::factory('/path/to/config.php');

// Get the client from the service locator by namespace
$s3Client = $aws->get('s3');
```

A simple configuration file should look something like this:

```
<?php return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
                'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
                'region' => 'us-west-2'
            )
        )
    )
);
```

For more information about configuration files, please see *Configuring the SDK*.

Performing service operations

You can perform a service **operation** by calling the method of the same name on the client object. For example, to perform the Amazon DynamoDB DescribeTable operation, you must call the `Aws\DynamoDb\DynamoDbClient::describeTable()` method. Operation methods, like `describeTable()`, all accept a single argument that is an associative array of values representing the parameters to the operation. The structure of this array is defined for each operation in the SDK's API Documentation (e.g., see the API docs for describeTable()).

```
$result = $dynamoDbClient->describeTable(array(
    'TableName' => 'YourTableName',
));
```

To learn about performing operations in more detail, including using command objects, see *Command Objects*.

Working with modeled responses

The result of a performing an operation is what we refer to as a **modeled response**. Instead of returning the raw XML or JSON data, the SDK will coerce the data into an associative array and normalize some aspects of the data based on its knowledge of the specific service and the underlying response structure.

The actual value returned is a Model (`Guzzle\Service\Resource\Model`) object. The Model class is a part of the SDK's underlying Guzzle library, but you do not need to know anything about Guzzle to use your operation results. The Model object contains the data from the response and can be used like an array (e.g., `$result['Table']`). It also has convenience methods like `get()`, `getPath()`, and `toArray()`. The contents of the modeled response depend on the operation that was executed and are documented in the API docs for each operation (e.g., see the *Returns* section in the API docs for the DynamoDB DescribeTable operation).

```
$result = $dynamoDbClient->describeTable(array(
    'TableName' => 'YourTableName',
));

// Get a specific value from the result
$table = $result['Table'];
if ($table && isset($table['TableStatus'])) {
    echo $table['TableStatus'];
}
//> ACTIVE

// Get nested values from the result easily
echo $result->getPath('Table/TableStatus');
//> ACTIVE

// Convert the Model to a plain array
var_export($result->toArray());
//> array ( 'Table' => array ( 'AttributeDefinitions' => array ( ... ) ... ) ... )
```

To learn more about how to work with modeled responses, read the detailed guide to *Modeled Responses*.

Detecting and handling errors

When you preform an operation, and it succeeds, it will return a modeled response. If there was an error with the request, then an exception is thrown. For this reason, you should use `try/catch` blocks around your operations if you need to handle errors in your code. The SDK throws service-specific exceptions when a server-side error occurs.

In the following example, the `Aws\S3\S3Client` is used. If there is an error, the exception thrown will be of the type: `Aws\S3\Exception\S3Exception`.

```
try {
    $s3Client->createBucket(array(
        'Bucket' => 'my-bucket'
    ));
} catch (\Aws\S3\Exception\S3Exception $e) {
    // The bucket couldn't be created
    echo $e->getMessage();
}
```

Exceptions thrown by the SDK like this all extend the `ServiceResponseException` class (see the API docs), which has some custom methods that might help you discover what went wrong.

Waiters

One of the higher-level abstractions provided by the SDK are **waiters**. Waiters help make it easier to work with *eventually consistent* systems by providing an easy way to wait until a resource enters into a particular state by polling the resource. You can find a list of the waiters supported by a client by viewing the API Documentation of a service client. Any method with a name starting with "`waitUntil`" will create and invoke a waiter.

In the following example, the Amazon S3 Client is used to create a bucket. Then the waiter method is used to wait until the bucket exists.

```
// Create a bucket
$s3Client->createBucket(array('Bucket' => 'my-bucket'));

// Wait until the created bucket is available
$s3Client->waitUntilBucketExists(array('Bucket' => 'my-bucket'));
```

To learn more about how to use and configure waiters, please read the detailed guide to *Waiters*.

Iterators

Some AWS operations return truncated results that require subsequent requests in order to retrieve the entire result set. The subsequent requests typically require pagination tokens or markers from the previous request in order to retrieve the next set of results. Working with these tokens can be cumbersome, since you must manually keep track of them, and the API for each service may differ in how it uses them.

The AWS SDK for PHP has a feature called **iterators** that allow you to retrieve an *entire* result set without manually handling pagination tokens or markers. The iterators in the SDK implement PHP's `Iterator` interface, which allows you to easily enumerate or iterate through resources from a result set with `foreach`.

You can find a list of the iterators supported by a client by viewing the docblock of a client. Any `@method` tag that has a name that looks like "`get[…]Iterator`" will return an iterator. For example, the following code uses the `getListObjectsIterator()` method of the S3 client object to create an iterator for objects in a bucket.

```
$iterator = $client->getListObjectsIterator(array('Bucket' => 'my-bucket'));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

To learn more about how to use and configure iterators, please read the detailed guide to *Iterators*.

Migration Guide

This guide shows how to migrate your code to use the new AWS SDK for PHP and how the new SDK differs from the AWS SDK for PHP - Version 1.

Introduction

The PHP language and community have evolved significantly over the past few years. Since the inception of the AWS SDK for PHP, PHP has gone through two major version changes (versions 5.3 and 5.4) and many in the PHP community have unified behind the recommendations of the PHP Framework Interop Group. Consequently, we decided to make breaking changes to the SDK in order to align with the more modern patterns used in the PHP community.

For the new release, we rewrote the SDK from the ground up to address popular customer requests. The new SDK is built on top of the Guzzle HTTP client framework, which provides increased performance and enables event-driven customization. We also introduced high-level abstractions to make programming common tasks easy. The SDK is compatible with PHP 5.3.3 and newer, and follows the PSR-0 standard for namespaces and autoloading.

Which Services are Supported?

The AWS SDK for PHP supports all of the AWS services supported by Version 1 of the SDK and more, including Amazon Route 53, Amazon Glacier, and AWS Direct Connect. See the AWS SDK for PHP website for the full list of services supported by the SDK. Be sure to watch or star our AWS SDK for PHP GitHub repository to stay up-to-date with the latest changes.

What's New?

- PHP 5.3 namespaces
- Follows PSR-0, PSR-1, and PSR-2 standards
- Built on Guzzle and utilizes the Guzzle feature set
- Persistent connection management for both serial and parallel requests
- Event hooks (via Symfony2 EventDispatcher) for event-driven, custom behavior
- Request and response entity bodies are stored in `php://temp` streams to reduce memory usage
- Transient networking and cURL failures are automatically retried using truncated exponential backoff
- Plug-ins for over-the-wire logging and response caching
- "Waiter" objects that allow you to poll a resource until it is in a desired state
- Resource iterator objects for easily iterating over paginated responses
- Service-specific sets of exceptions
- Modeled responses with a simpler interface
- Grouped constants (Enums) for service parameter options
- Flexible request batching system
- Service builder/container that supports easy configuration and dependency injection
- Full unit test suite with extensive code coverage
- Composer support (including PSR-0 compliance) for installing and autoloading SDK dependencies
- Phing `build.xml` for installing dev tools, driving testing, and producing `.phar` files
- Fast Amazon DynamoDB batch PutItem and DeleteItem system
- Multipart upload system for Amazon Simple Storage Service (Amazon S3) and Amazon Glacier that can be paused and resumed
- Redesigned DynamoDB Session Handler with smarter writing and garbage collection
- Improved multi-region support

What's Different?

Architecture

The new SDK is built on top of Guzzle and inherits its features and conventions. Every AWS service client extends the Guzzle client, defining operations through a service description file. The SDK has a much more robust and flexible object-oriented architecture, including the use of design patterns, event dispatching and dependency injection. As a result, many of the classes and methods from the previous SDK have been changed.

Project Dependencies

Unlike the Version 1 of the SDK, the new SDK does not pre-package all of its dependencies in the repository. Dependencies are best resolved and autoloaded via Composer. However, when installing the SDK via the downloadable phar, the dependencies are resolved for you.

Namespaces

The SDK's directory structure and namespaces are organized according to PSR-0 standards, making the SDK inherently modular. The `Aws\Common` namespace contains the core code of the SDK, and each service client is contained in its own separate namespace (e.g., `Aws\DynamoDb`).

Coding Standards

The SDK adopts the PSR standards produced by the PHP Framework Interop Group. An immediately noticeable change is that all method names are now named using lower camel-case (e.g., `putObject` instead of `put_object`).

Required Regions

The region must be provided to instantiate a client (except in the case where the service has a single endpoint like Amazon CloudFront). The AWS region you select may affect both your performance and costs.

Client Factories

Factory methods instantiate service clients and do the work of setting up the signature, exponential backoff settings, exception handler, and so forth. At a minimum you must provide your access key, secret key, and region to the client factory, but there are many other settings you can use to customize the client behavior.

```
$dynamodb = Aws\DynamoDb\DynamoDbClient::factory(array(
    'key'    => 'your-aws-access-key-id',
    'secret' => 'your-aws-secret-access-key',
    'region' => 'us-west-2',
));
```

Configuration

A global configuration file can be used to inject credentials into clients automatically via the service builder. The service builder acts as a dependency injection container for the service clients. (**Note:** The SDK does not automatically attempt to load the configuration file like in Version 1 of the SDK.)

```
$aws = Aws\Common\Aws::factory('/path/to/custom/config.php');
$s3 = $aws->get('s3');
```

This technique is the preferred way for instantiating service clients. Your `config.php` might look similar to the following:

```
<?php
return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'your-aws-access-key-id',
                'secret' => 'your-aws-secret-access-key',
                'region' => 'us-west-2'
            )
        )
    )
);
```

The line that says `'includes' => array('_aws')` includes the default configuration file packaged with the SDK. This sets up all of the service clients for you so you can retrieve them by name with the `get()` method of the service builder.

Service Operations

Executing operations in the new SDK is similar to how it was in the previous SDK, with two main differences. First, operations follow the lower camel-case naming convention. Second, a single array parameter is used to pass in all of the operation options. The following examples show the Amazon S3 `PutObject` operation performed in each SDK:

```
// Previous SDK - PutObject operation
$s3->create_object('bucket-name', 'object-key.txt', array(
    'body' => 'lorem ipsum'
));
```

```
// New SDK - PutObject operation
$result = $s3->putObject(array(
    'Bucket' => 'bucket-name',
    'Key'    => 'object-key.txt',
    'Body'   => 'lorem ipsum'
));
```

In the new SDK, the `putObject()` method doesn't actually exist as a method on the client. It is implemented using the `__call()` magic method of the client and acts as a shortcut to instantiate a command, execute the command, and retrieve the result.

A `Command` object encapsulates the request and response of the call to AWS. From the `Command` object, you can call the `getResult()` method (as in the preceding example) to retrieve the parsed result, or you can call the `getResponse()` method to retrieve data about the response (e.g., the status code or the raw response).

The `Command` object can also be useful when you want to manipulate the command before execution or need to execute several commands in parallel. The following is an example of the same `PutObject` operation using the command syntax:

```
$command = $s3->getCommand('PutObject', array(
    'Bucket' => 'bucket-name',
    'Key'    => 'object-key.txt',
    'Body'   => 'lorem ipsum'
));
$result = $command->getResult();
```

Or you can use the chainable `set()` method on the `Command` object:

```
$result = $s3->getCommand('PutObject')
    ->set('Bucket', 'bucket-name')
    ->set('Key', 'object-key.txt')
    ->set('Body', 'lorem ipsum')
    ->getResult();
```

Responses

The format of responses has changed. Responses are no longer instances of the `CFResponse` object. The `Command` object (as seen in the preceding section) of the new SDK encapsulates the request and response, and is the object from which to retrieve the results.

```
// Previous SDK
// Execute the operation and get the CFResponse object
$response = $s3->list_tables();
// Get the parsed response body as a SimpleXMLElement
$result = $response->body;

// New SDK
// Executes the operation and gets the response in an array-like object
$result = $s3->listTables();
```

The new syntax is similar, but a few fundamental differences exist between responses in the previous SDK and this version:

The new SDK represents parsed responses (i.e., the results) as Guzzle `Model` objects instead of `CFSimpleXML` objects as in the prior version. These Model objects are easy to work with since they act like arrays. They also have helpful built-in features such as mapping and filtering. The content of the results will also look different n this version of the SDK. The SDK marshals responses into the models and then transforms them into more convenient structures based on the service description. The API documentation details the response of all operations.

Exceptions

The new SDK uses exceptions to communicate errors and bad responses.

Instead of relying on the `CFResponse::isOK()` method of the previous SDK to determine if an operation is successful, the new SDK throws exceptions when the operation is *not* successful. Therefore, you can assume success if there was no exception thrown, but you will need to add `try...catch` logic to your application code in order to handle potential errors. The following is an example of how to handle the response of an Amazon DynamoDB `DescribeTable` call in the new SDK:

```
$tableName = 'my-table';
try {
    $result = $dynamoDb->describeTable(array('TableName' => $tableName));

    printf('The provisioned throughput for table "%s" is %d RCUs and %d WCUs.',
        $tableName,
        $result->getPath('Table/ProvisionedThroughput/ReadCapacityUnits'),
        $result->getPath('Table/ProvisionedThroughput/WriteCapacityUnits')
    );
} catch (Aws\DynamoDb\Exception\DynamoDbException $e) {
    echo "Error describing table {$tableName}";
}
```

You can get the Guzzle response object back from the command. This is helpful if you need to retrieve the status code, additional data from the headers, or the raw response body.

```
$command = $dynamoDb->getCommand('DescribeTable', array('TableName' => $tableName));
$statusCode = $command->getResponse()->getStatusCode();
```

You can also get the response object and status code from the exception if one is thrown.

```
try {
    $command = $dynamoDb->getCommand('DescribeTable', array(
        'TableName' => $tableName
    ));
    $statusCode = $command->getResponse()->getStatusCode();
} catch (Aws\DynamoDb\Exception\DynamoDbException $e) {
    $statusCode = $e->getResponse()->getStatusCode();
}
```

Iterators

The SDK provides iterator classes that make it easier to traverse results from list and describe type operations. Instead of having to code solutions that perform multiple requests in a loop and keep track of tokens or markers, the iterator classes do that for you. You can simply foreach over the iterator:

```
$objects = $s3->getIterator('ListObjects', array(
    'Bucket' => 'my-bucket-name'
));

foreach ($objects as $object) {
    echo $object['Key'] . PHP_EOL;
}
```

Comparing Code Samples from Both SDKs

Example 1 - Amazon S3 ListParts Operation

From Version 1 of the SDK

```
<?php

require '/path/to/sdk.class.php';
require '/path/to/config.inc.php';
```

```php
$s3 = new AmazonS3();

$response = $s3->list_parts('my-bucket-name', 'my-object-key', 'my-upload-id', array(
    'max-parts' => 10
));

if ($response->isOK())
{
    // Loop through and display the part numbers
    foreach ($response->body->Part as $part) {
        echo "{$part->PartNumber}\n";
    }
}
else
{
    echo "Error during S3 ListParts operation.\n";
}
```

From Version 2 of the SDK

```php
<?php

require '/path/to/vendor/autoload.php';

use Aws\Common\Aws;
use Aws\S3\Exception\S3Exception;

$aws = Aws::factory('/path/to/config.php');
$s3 = $aws->get('s3');

try {
    $result = $s3->listParts(array(
        'Bucket'   => 'my-bucket-name',
        'Key'      => 'my-object-key',
        'UploadId' => 'my-upload-id',
        'MaxParts' => 10
    ));

    // Loop through and display the part numbers
    foreach ($result['Part'] as $part) {
        echo "{$part[PartNumber]}\n";
    }
} catch (S3Exception $e) {
    echo "Error during S3 ListParts operation.\n";
}
```

Example 2 - Amazon DynamoDB Scan Operation

From Version 1 of the SDK

```php
<?php

require '/path/to/sdk.class.php';
require '/path/to/config.inc.php';

$dynamo_db = new AmazonDynamoDB();

$start_key = null;
```

```php
$people = array();

// Perform as many Scan operations as needed to acquire all the names of people
// that are 16 or older
do
{
    // Setup the parameters for the DynamoDB Scan operation
    $params = array(
        'TableName'       => 'people',
        'AttributesToGet' => array('id', 'age', 'name'),
        'ScanFilter'      => array(
            'age' => array(
                'ComparisonOperator' =>
                    AmazonDynamoDB::CONDITION_GREATER_THAN_OR_EQUAL,
                'AttributeValueList' => array(
                    array(AmazonDynamoDB::TYPE_NUMBER => '16')
                )
            ),
        )
    );

    // Add the exclusive start key parameter if needed
    if ($start_key)
    {
        $params['ExclusiveStartKey'] = array(
            'HashKeyElement' => array(
                AmazonDynamoDB::TYPE_STRING => $start_key
            )
        );

        $start_key = null;
    }

    // Perform the Scan operation and get the response
    $response = $dynamo_db->scan($params);

    // If the response succeeded, get the results
    if ($response->isOK())
    {
        foreach ($response->body->Items as $item)
        {
            $people[] = (string) $item->name->{AmazonDynamoDB::TYPE_STRING};
        }

        // Get the last evaluated key if it is provided
        if ($response->body->LastEvaluatedKey)
        {
            $start_key = (string) $response->body
                ->LastEvaluatedKey
                ->HashKeyElement
                ->{AmazonDynamoDB::TYPE_STRING};
        }
    }
    else
    {
        // Throw an exception if the response was not OK (200-level)
        throw new DynamoDB_Exception('DynamoDB Scan operation failed.');
    }
}
while ($start_key);
```

```
print_r($people);
```

From Version 2 of the SDK

```
<?php

require '/path/to/vendor/autoload.php';

use Aws\Common\Aws;
use Aws\DynamoDb\Enum\ComparisonOperator;
use Aws\DynamoDb\Enum\Type;

$aws = Aws::factory('/path/to/config.php');
$dynamodb = $aws->get('dynamodb');

// Create a ScanIterator and setup the parameters for the DynamoDB Scan operation
$scan = $dynamodb->getIterator('Scan', array(
    'TableName'       => 'people',
    'AttributesToGet' => array('id', 'age', 'name'),
    'ScanFilter'      => array(
        'age' => array(
            'ComparisonOperator' => ComparisonOperator::GE,
            'AttributeValueList' => array(
                array(Type::NUMBER => '16')
            )
        ),
    )
));

// Perform as many Scan operations as needed to acquire all the names of people
// that are 16 or older
$people = array();
foreach ($scan as $item) {
    $people[] = $item['name'][Type::STRING];
}

print_r($people);
```

Side-by-side Guide

This guide helps you install, configure, and run Version 1 and Version 2 of the AWS SDK for PHP side-by-side within the same application or project. Please see the *Migration Guide* for more information on migrating code from the original AWS SDK for PHP to Version 2.

Since Version 2 of the AWS SDK for PHP now supports all of the AWS services supported by Version 1 (and more), it is recommended that you should begin migrating your code to use Version 2 of the SDK. Using both SDKs side-by-side may be helpful if your use case requires you to migrate only sections of your code at a time.

Installing and Including the SDKs

To install and include the SDKs in your project, you must first choose whether or not to use Composer.

Using Composer

Using Composer is the recommended way to install both versions of the AWS SDK for PHP. Composer is a dependency management tool for PHP that allows you to declare the dependencies your project requres and installs them into your project. In order to simultaneously use both versions of the SDK in the same project through Composer, you must do the following:

1. Add both of the SDKs as dependencies in your project's `composer.json` file.

```
{
    "require": {
        "aws/aws-sdk-php": "*",
        "amazonwebservices/aws-sdk-for-php": "*"
    }
}
```

   Note: Consider tightening your dependencies to a known version when deploying mission critical applications (e.g., `2.0.*`).

2. Download and install Composer.

```
curl -s "http://getcomposer.org/installer" | php
```

3. Install your dependencies.

```
php composer.phar install
```

4. Require Composer's autoloader.

   Composer also prepares an autoload file that's capable of autoloading all of the classes in any of the libraries that it downloads. To use it, just add the following line to your code's bootstrap process.

```
require '/path/to/sdk/vendor/autoload.php';
```

You can find out more on how to install Composer, configure autoloading, and other best-practices for defining dependencies at getcomposer.org.

Without Composer

Without Composer, you must manage your own project dependencies.

1. Download both of the SDKs (via PEAR, GitHub, or the AWS website) into a location accessible by your project. Make certain to use the pre-packaged `aws.phar` file, which includes all of the dependencies for the AWS SDK for PHP.
2. In your code's bootstrap process, you need to explicitly require the bootstrap file from Version 1 of the SDK and the `aws.phar` file containing Version 2 of the SDK:

```
// Include each of the SDK's bootstrap files to setup autoloading
require '/path/to/sdk.class.php'; // Load the Version 1 bootstrap file
require '/path/to/aws.phar';      // Load the Version 2 pre-packaged phar file
```

Configuring and Instantiating the SDKs

How you configure and instantiate the SDKs is determined by whether or not you are using the service builder (`Aws\Common\Aws` class).

Instantiating Clients via the Service Builder

The service builder (`Aws\Common\Aws class`) in the AWS SDK for PHP enables configuring all service clients with the same credentials. It also accepts additional settings for some or all of the clients. The service builder functionality is inherited from the Guzzle project.

You can pass the service builder a configuration file containing your credentials and other settings. It will then inject these into all of the service clients your code instantiates. For more information about the configuration file, please read the *Configuring the SDK* section of the guide. When using both SDKs side-by-side, your configuration file must include the following line:

```
'includes' => array('_sdk1'),
```

This will automatically set up the service clients from Version 1 of the SDK making them accessible through the service builder by keys such as `v1.s3` and `v1.cloudformation`. Here is an example configuration file that includes referencing the Version 1 of the SDK:

```
<?php return array(
    'includes' => array('_sdk1'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'your-aws-access-key-id',
                'secret' => 'your-aws-secret-access-key',
                'region' => 'us-west-2'
            )
        )
    )
);
```

Your code must instantiate the service builder through its factory method by passing in the path of the configuration file. Your code then retrieves instances of the specific service clients from the returned builder object.

```
use Aws\Common\Aws;

// Instantiate the service builder
$aws = Aws::factory('/path/to/your/config.php');

// Instantiate S3 clients via the service builder
$s3v1 = $aws->get('v1.s3');  // All Version 1 clients are prefixed with "v1."
$s3v2 = $aws->get('s3');
```

Instantiating Clients via Client Factories

Your code can instantiate service clients using their respective `factory()` methods by passing in an array of configuration data, including your credentials. The `factory()` will work for clients in either versions of the SDK.

```
use Aws\S3\S3Client;

// Create an array of configuration options
$config = array(
    'key'    => 'your-aws-access-key-id',
    'secret' => 'your-aws-secret-access-key',
);

// Instantiate Amazon S3 clients from both SDKs via their factory methods
$s3v1 = AmazonS3::factory($config);
$s3v2 = S3Client::factory($config);
```

Optionally, you could alias the classes to make it clearer which version of the SDK they are from.

```
use AmazonS3 as S3ClientV1;
use Aws\S3\S3Client as S3ClientV2;

$config = array(
    'key'    => 'your-aws-access-key-id',
    'secret' => 'your-aws-secret-access-key',
);

$s3v1 = S3ClientV1::factory($config);
$s3v2 = S3ClientV2::factory($config);
```

Complete Examples

The following two examples fully demonstrate including, configuring, instantiating, and using both SDKs side-by-side. These examples adopt the recommended practices of using Composer and the service builder.

Example 1 - Dual Amazon S3 Clients

This example demonstrates using an Amazon S3 client from the AWS SDK for PHP working side-by-side with an Amazon S3 client from the first PHP SDK.

```php
<?php

require 'vendor/autoload.php';

$aws = Aws\Common\Aws::factory('/path/to/config.json');

$s3v1 = $aws->get('v1.s3');
$s3v2 = $aws->get('s3');

echo "ListBuckets with SDK Version 1:\n";
echo "------------------------------\n";
$response = $s3v1->listBuckets();
if ($response->isOK()) {
    foreach ($response->body->Buckets->Bucket as $bucket) {
        echo "- {$bucket->Name}\n";
    }
} else {
    echo "Request failed.\n";
}
echo "\n";

echo "ListBuckets with SDK Version 2:\n";
echo "------------------------------\n";
try {
    $result = $s3v2->listBuckets();
    foreach ($result['Buckets'] as $bucket) {
        echo "- {$bucket['Name']}\n";
    }
} catch (Aws\S3\Exception\S3Exception $e) {
    echo "Request failed.\n";
}
echo "\n";
```

Example 2 - Amazon DynamoDB and Amazon SNS Clients

This example shows how the AWS SDK for PHP DynamoDB client works together with the SNS client from the original SDK. For this example, an ice cream parlor publishes a daily message (via SNS) containing its "flavors of the day" to subscribers. First, it retrieves the flavors of the day from its DynamoDB database using the AWS SDK for PHP DynamoDB client. It then uses the SNS client from the first SDK to publish a message to its SNS topic.

```php
<?php

require 'vendor/autoload.php';

$aws = Aws\Common\Aws::factory('/path/to/config.php');

// Instantiate the clients
$ddb = $aws->get('dynamodb');
$sns = $aws->get('v1.sns');
$sns->set_region(AmazonSNS::REGION_US_W2);

// Get today's flavors from DynamoDB using Version 2 of the SDK
$date = new DateTime();
$flavors = $ddb->getItem(array(
    'TableName' => 'flavors-of-the-day',
    'Key' => array(
        'HashKeyElement'  => array('N' => $date->format('n')),
```

```
        'RangeKeyElement' => array('N' => $date->format('j'))
    )
))->getResult()->getPath('Item/flavors/SS');

// Generate the message
$today = $date->format('l, F jS');
$message = "It's {$today}, and here are our flavors of the day:\n";
foreach ($flavors as $flavor) {
    $message .= "- {$flavor}\n";
}
$message .= "\nCome visit Mr. Foo\'s Ice Cream Parlor on 5th and Pine!\n";
echo "{$message}\n";

// Send today's flavors to subscribers using Version 1 of the SDK
$response = $sns->publish('flavors-of-the-day-sns-topic', $message, array(
    'Subject' => 'Flavors of the Day - Mr. Foo\'s Ice Cream Parlor'
));
if ($response->isOK()) {
    echo "Sent the flavors of the day to your subscribers.\n";
} else {
    echo "There was an error sending the flavors of the day to your subscribers.\n";
}
```

Final Notes

Remember that **instantiating clients from the original SDK using the service builder from AWS SDK for PHP does not change how those clients work**. For example, notice the differences in response handling between SDK versions. For a full list of differences between the versions, please see the *Migration Guide*.

For more information about using the original version of the SDK, please see the Version 1 API Documentation and the Version 1 SDK README.

Providing Credentials to the SDK

Introduction

In order to authenticate requests, AWS services require you to provide your AWS access keys, also known as your AWS **access key ID** and **secret access key**. In the AWS SDK for PHP, these access keys are often referred to collectively as your **credentials**. This guide demonstrates how to provide your credentials to the AWS SDK for SDK.

There are many ways to provide credentials:

1. *Using credentials from environment variables*
2. *Using IAM roles for Amazon EC2 instances*
3. *Using a configuration file with the service builder*
4. *Passing credentials into a client factory method*
5. *Setting credentials after instantiation*
6. *Using temporary credentials from AWS STS*

Which technique should you choose?

The technique that you use to provide credentials to the SDK for your application is entirely up to you. Please read each section on this page to determine what is the best fit for you. What you choose will depend on many different factors, including:

- The environment you are operating in (e.g., development, testing, production)
- The host of your application (e.g., localhost, Amazon EC2, third-party server)

- How many sets of credentials you are using
- The type of project you are developing (e.g., application, CLI, library)
- How often you rotate your credentials
- If you rely on temporary or federated credentials
- Your deployment process
- Your application framework

Regardless of the technique used, it is encouraged that you follow the IAM Best Practices when managing your credentials, including the recommendation to not use your AWS account's root credentials. Instead, create separate IAM users with their own access keys for each project, and tailor the permissions of the users specific to those projects.

In general, it is recommended that you use IAM roles when running your application on Amazon EC2 and use environment variables elsewhere.

Using credentials from environment variables

If you do not provide credentials to a client object at the time of its instantiation (e.g., via the client's factory method or via a service builder configuration), the SDK will attempt to find credentials in your environment when you call your first operation. The SDK will use the `$_SERVER` superglobal and `getenv()` function to look for the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_KEY` environment variables. These credentials are often called **environment credentials**.

If you are hosting your application on AWS Elastic Beanstalk, you can set the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_KEY` environment variables on the AWS Elastic Beanstalk console so that the SDK can use those credentials automatically.

Using IAM roles for Amazon EC2 instances

Using IAM roles is the preferred technique for providing credentials to applications running on Amazon EC2. IAM roles remove the need to worry about credential management from your application. They allow an instance to "assume" a role by retrieving temporary credentials from the EC2 instance's metadata server. These temporary credentials, often referred to as **instance profile credentials**, allow access to the actions and resources that the role's policy allows.

When launching an EC2 instance, you can choose to associate it with an IAM role. Any application running on that EC2 instance is then allowed to assume the associated role. Amazon EC2 handles all the legwork of securely authenticating instances to the IAM service to assume the role and periodically refreshing the retrieved role credentials, keeping your application secure with almost no work on your part.

If you do not explicitly provide credentials to the client object and no environment variable credentials are available, the SDK attempts to retrieve instance profile credentials from an Amazon EC2 instance metadata server. These credentials are available only when running on Amazon EC2 instances that have been configured with an IAM role.

Note

Instance profile credentials and other temporary credentials generated by the AWS Security Token Service (AWS STS) are not supported by every service. Please check if the service you are using supports temporary credentials by reading AWS Services that Support AWS STS.

For more information, see IAM Roles for Amazon EC2.

Caching IAM role credentials

While using IAM role credentials is the preferred method for providing credentials to an application running on an Amazon EC2 instance, the roundtrip from the application to the instance metadata server on each request can introduce latency. In these situations, you might find that utilizing a caching layer on top of your IAM role credentials can eliminate the introduced latency.

The easiest way to add a cache to your IAM role credentials is to specify a credentials cache using the `credentials.cache` option in a client's factory method or in a service builder configuration file. The `credentials.cache` configuration setting should be set to an object that implements Guzzle's `Guzzle\Cache\CacheAdapterInterface` (see Guzzle cache adapters). This interface provides an abstraction layer over various cache backends, including Doctrine Cache, Zend Framework 2 cache, etc.

```
<?php

require 'vendor/autoload.php';

use Doctrine\Common\Cache\FilesystemCache;
use Guzzle\Cache\DoctrineCacheAdapter;

// Create a cache adapter that stores data on the filesystem
$cacheAdapter = new DoctrineCacheAdapter(new FilesystemCache('/tmp/cache'));

// Provide a credentials.cache to cache credentials to the file system
$s3Client = Aws\S3\S3Client::factory(array(
    'credentials.cache' => $cacheAdapter
));
```

In the preceding example, the addition of `credentials.cache` causes credentials to be cached to the local filesystem using Doctrine's caching system. Every request that uses this cache adapter first checks if the credentials are in the cache. If the credentials are found in the cache, the client then ensures that the credentials are not expired. In the event that cached credentials become expired, the client automatically refreshes the credentials on the next request and populates the cache with the updated credentials.

A credentials cache can also be used in a service builder configuration:

```
<?php

// File saved as /path/to/custom/config.php

use Doctrine\Common\Cache\FilesystemCache;
use Guzzle\Cache\DoctrineCacheAdapter;

$cacheAdapter = new DoctrineCacheAdapter(new FilesystemCache('/tmp/cache'));

return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'credentials.cache' => $cacheAdapter
            )
        )
    )
);
```

If you were to use the above configuration file with a service builder, then all of the clients created through the service builder would utilize a shared credentials cache object.

Setting credentials explicitly in your code

The SDK allows you to explicitly set your credentials in your project in a few different ways. These techniques are useful for rapid development, integrating with existing configurations systems (e.g., your PHP framework of choice), and handling *temporary credentials*. However, **be careful to not hard-code your credentials** inside of your applications. Hard-coding your credentials can be dangerous, because it is easy to accidentally commit your credentials into an SCM repository, potentially exposing your credentials to more people than intended. It can also make it difficult to rotate credentials in the future.

Using a configuration file with the service builder

The SDK provides a service builder that can be used to share configuration values across multiple clients. The service builder allows you to specify default configuration values (e.g., credentials and regions) that are used by every client. The service builder is configured using either JSON configuration files or PHP scripts that return an array.

The following is an example of a configuration script that returns an array of configuration data that can be used by the service builder:

```php
<?php

return array(
    // Bootstrap the configuration file with AWS specific features
    'includes' => array('_aws'),
    'services' => array(
        // All AWS clients extend from 'default_settings'. Here we are
        // overriding 'default_settings' with our default credentials and
        // providing a default region setting.
        'default_settings' => array(
            'params' => array(
                'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
                'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
                'region' => 'us-west-1'
            )
        )
    )
);
```

After creating and saving the configuration file, you need to instantiate a service builder.

```php
<?php

use Aws\Common\Aws;

// Create the AWS service builder, providing the path to the config file
$aws = Aws::factory('/path/to/custom/config.php');
```

At this point, you can now create clients using the `get()` method of the `Aws` object:

```
$s3Client = $aws->get('s3');
```

Passing credentials into a client factory method

A simple way to specify your credentials is by injecting them directly into the factory method when instantiating the client object.

```php
<?php

use Aws\S3\S3Client;

// Instantiate the S3 client with your AWS credentials
$s3Client = S3Client::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
));
```

In some cases, you may already have an instance of a `Credentials` object. You can use this instead of specifying your access keys separately.

```php
<?php

use Aws\S3\S3Client;
use Aws\Common\Credentials\Credentials;

$credentials = new Credentials('YOUR_ACCESS_KEY', 'YOUR_SECRET_KEY');
```

```
// Instantiate the S3 client with your AWS credentials
$s3Client = S3Client::factory(array(
    'credentials' => $credentials
));
```

You may also want to read the section in the Getting Started Guide about *using a client's factory method* for more details.

Setting credentials after instantiation

At any time after instantiating the client, you can set the credentials the client should use with the `setCredentials()` method.

```
<?php

use Aws\S3\S3Client;
use Aws\Common\Credentials\Credentials

$s3Client = S3Client::factory();

$credentials = new Credentials('YOUR_ACCESS_KEY', 'YOUR_SECRET_KEY');
$s3Client->setCredentials($credentials);
```

This can be used to change the credentials, set temporary credentials, refresh expired credentials, etc.

Using the `setCredentials()` method will also trigger a `client.credentials_changed` event, so you can program other parts of your application to react to the change. To do this, you just need to add a listener to the client object.

```
use Aws\S3\S3Client;
use Aws\Common\Credentials\Credentials

// Create 2 sets of credentials
$credentials1 = new Credentials('ACCESS_KEY_1', 'SECRET_KEY_1');
$credentials2 = new Credentials('ACCESS_KEY_2', 'SECRET_KEY_2');

// Instantiate the client with the first credential set
$s3Client = S3Client::factory(array('credentials' => $credentials1));

// Get the event dispatcher and register a listener for the credential change
$dispatcher = $s3Client->getEventDispatcher();
$dispatcher->addListener('client.credentials_changed', function ($event) {
    $formerAccessKey = $event['former_credentials']->getAccessKey();
    $currentAccessKey = $event['credentials']->getAccessKey();
    echo "Access key has changed from {$formerAccessKey} to {$currentAccessKey}.\n";
});

// Change the credentials to the second set to trigger the event
$s3Client->setCredentials($credentials2);
//> Access key has changed from ACCESS_KEY_1 to ACCESS_KEY_2.
```

Using temporary credentials from AWS STS

AWS Security Token Service (AWS STS) enables you to request limited-privilege, **temporary credentials** for AWS IAM users or for users that you authenticate via identity federation. One common use case for using temporary credentials is to grant mobile or client-side applications access to AWS resources by authenticating users through third-party identity providers (read more about Web Identity Federation).

Note

Temporary credentials generated by AWS STS are not supported by every service. Please check if the service you are using supports temporary credentials by reading AWS Services that Support AWS STS.

Getting temporary credentials

AWS STS has several operations that return temporary credentials, but the `GetSessionToken` operation is the simplest for demonstration purposes. Assuming you have an instance of `Aws\Sts\StsClient` stored in the `$stsClient` variable, this is how you call it:

```
$result = $stsClient->getSessionToken();
```

The result for `GetSessionToken` and the other AWS STS operations always contains a `'Credentials'` value. If you print the result (e.g., `print_r($result)`), it looks like the following:

```
Array
(
    ...
    [Credentials] => Array
    (
        [SessionToken] => '<base64 encoded session token value>'
        [SecretAccessKey] => '<temporary secret access key value>'
        [Expiration] => 2013-11-01T01:57:52Z
        [AccessKeyId] => '<temporary access key value>'
    )
    ...
)
```

Providing temporary credentials to the SDK

You can use temporary credentials with another AWS client by instantiating the client and passing in the values received from AWS STS directly.

```
use Aws\S3\S3Client;

$result = $stsClient->getSessionToken();

$s3Client = S3Client::factory(array(
    'key'    => $result['Credentials']['AccessKeyId'],
    'secret' => $result['Credentials']['SecretAccessKey'],
    'token'  => $result['Credentials']['SessionToken'],
));
```

You can also construct a `Credentials` object and use that when instantiating the client.

```
use Aws\Common\Credentials\Credentials;
use Aws\S3\S3Client;

$result = $stsClient->getSessionToken();

$credentials = new Credentials(
    $result['Credentials']['AccessKeyId'],
    $result['Credentials']['SecretAccessKey'],
    $result['Credentials']['SessionToken']
);

$s3Client = S3Client::factory(array('credentials' => $credentials));
```

However, the *best* way to provide temporary credentials is to use the `createCredentials()` helper method included with the `StsClient`. This method extracts the data from an AWS STS result and creates the `Credentials` object for you.

```
$result = $stsClient->getSessionToken();
$credentials = $stsClient->createCredentials($result);

$s3Client = S3Client::factory(array('credentials' => $credentials));
```

You can also use the same technique when setting credentials on an existing client object.

```
$credentials = $stsClient->createCredentials($stsClient->getSessionToken());
$s3Client->setCredentials($credentials);
```

For more information about why you might need to use temporary credentials in your application or project, see Scenarios for Granting Temporary Access in the AWS STS documentation.

Configuring the SDK

The AWS SDK for PHP can be configured in many ways to suit your needs. This guide highlights the use of configuration files with the service builder as well as individual client configuration options.

Configuration files

How configuration files work

When passing an array of parameters to the first argument of `Aws\Common\Aws::factory()`, the service builder loads the default `aws-config.php` file and merges the array of shared parameters into the default configuration.

Excerpt from `src/Aws/Common/Resources/aws-config.php`:

```
<?php return array(
    'class' => 'Aws\Common\Aws',
    'services' => array(
        'default_settings' => array(
            'params' => array()
        ),
        'autoscaling' => array(
            'alias'   => 'AutoScaling',
            'extends' => 'default_settings',
            'class'   => 'Aws\AutoScaling\AutoScalingClient'
        ),
        'cloudformation' => array(
            'alias'   => 'CloudFormation',
            'extends' => 'default_settings',
            'class'   => 'Aws\CloudFormation\CloudFormationClient'
        ),
    // ...
);
```

The `aws-config.php` file provides default configuration settings for associating client classes with service names. This file tells the `Aws\Common\Aws` service builder which class to instantiate when you reference a client by name.

You can supply your credentials and other configuration settings to the service builder so that each client is instantiated with those settings. To do this, pass an array of settings (including your `key` and `secret`) into the first argument of `Aws\Common\Aws::factory()`.

```
<?php

require 'vendor/autoload.php';

use Aws\Common\Aws;
```

```
$aws = Aws::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
    'region' => 'us-east-1',
));
```

Using a custom configuration file

You can use a custom configuration file that allows you to create custom named clients with pre-configured settings.

Let's say you want to use the default `aws-config.php` settings, but you want to supply your keys using a configuration file. Each service defined in the default configuration file extends from `default_settings` service. You can create a custom configuration file that extends the default configuration file and add credentials to the `default_settings` service:

```
<?php return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
                'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
                'region' => 'us-west-2'
            )
        )
    )
);
```

Make sure to include the `'includes' => array('_aws'),` line in your configuration file, because this extends the default configuration that makes all of the service clients available to the service builder. If this is missing, then you will get an exception when trying to retrieve a service client.

You can use your custom configuration file with the `Aws\Common\Aws` class by passing the full path to the configuration file in the first argument of the `factory()` method:

```
<?php

require 'vendor/autoload.php';

use Aws\Common\Aws;

$aws = Aws::factory('/path/to/custom/config.php');
```

You can create custom named services if you need to use multiple accounts with the same service:

```
<?php return array(
    'includes' => array('_aws'),
    'services' => array(
        'foo.dynamodb' => array(
            'extends' => 'dynamodb',
            'params'  => array(
                'key'    => 'your-aws-access-key-id-for-foo',
                'secret' => 'your-aws-secret-access-key-for-foo',
                'region' => 'us-west-2'
            )
        ),
        'bar.dynamodb' => array(
            'extends' => 'dynamodb',
            'params'  => array(
                'key'    => 'your-aws-access-key-id-for-bar',
                'secret' => 'your-aws-secret-access-key-for-bar',
                'region' => 'us-west-2'
            )
```

```
        )
    )
);
```

If you prefer JSON syntax, you can define your configuration in JSON format instead of PHP.

```
{
    "includes": ["_aws"],
    "services": {
        "default_settings": {
            "params": {
                "key": "your-aws-access-key-id",
                "secret": "your-aws-secret-access-key",
                "region": "us-west-2"
            }
        }
    }
}
```

For more information about writing custom configuration files, please see Using the Service Builder in the Guzzle documentation.

Client configuration options

Basic client configuration options include your `key` and `secret` credentials (see *Providing Credentials to the SDK*) and a `region` (see *Specifying a region*). For typical use cases, you will not need to provide more than these 3 options. The following represents all of the possible client configuration options for service clients in the SDK.

Credentials Options	
Options	**Description**
`key`	Your AWS access key ID. See AWS access keys.
`secret`	Your AWS secret access key. See AWS access keys.
`token`	An AWS security token to use with request authentication. Please note that not all services accept temporary credentials. See http://docs.aws.amazon.com/STS/latest/UsingSTS/UsingTokens.html.
`token.ttd`	The UNIX timestamp for when the provided credentials expire.
`credentials`	A credentials object (`Aws\Common\Credentials\CredentialsInterface`) can be provided instead explicit access keys and tokens.
`credentials.cache.key`	Optional custom cache key to use with the credentials.
`credentials.client`	Pass this option to specify a custom `Guzzle\Http\ClientInterface` to use if your credentials require a HTTP request (e.g. `RefreshableInstanceProfileCredentials`).

Endpoint and Signature Options	
Options	**Description**
`region`	Region name (e.g., 'us-east-1', 'us-west-1', 'us-west-2', 'eu-west-1', etc.). See *Specifying a region*.
`scheme`	URI Scheme of the base URL (e.g.. 'https', 'http') used when base_url is not supplied.
`base_url`	Allows you to specify a custom endpoint instead of have the SDK build one automatically from the region and scheme.
`signature`	Overrides the signature used by the client. Clients will always choose an appropriate default signature. However, it can be useful to override this with a custom setting. This can be set to "v4", "v3https", "v2" or an instance of `Aws\Common\Signature\SignatureInterface`.

`signature.service`	The signature service scope for Signature V4. See *Setting a custom endpoint*.
`signature.region`	The signature region scope for Signature V4. See *Setting a custom endpoint*.

Generic Client Options	
Options	**Description**
`ssl.certificate_authority`	Set to true to use the SDK bundled SSL certificate bundle (this is used by default), `'system'` to use the bundle on your system, a string pointing to a file to use a specific certificate file, a string pointing to a directory to use multiple certificates, or false to disable SSL validation (not recommended). When using the `aws.phar`, the bundled SSL certificate will be extracted to your system's temp folder, and each time a client is created an MD5 check will be performed to ensure the integrity of the certificate.
`curl.options`	Associative array of cURL options to apply to every request created by the client. If either the key or value of an entry in the array is a string, Guzzle will attempt to find a matching defined cURL constant automatically (e.g. `"CURLOPT_PROXY"` will be converted to the constant `CURLOPT_PROXY`).
`request.options`	Associative array of Guzzle request options to apply to every request created by the client.
`command.params`	An associative array of default options to set on each command created by the client.
`client.backoff.logger`	A `Guzzle\Log\LogAdapterInterface` object used to log backoff retries. Use `'debug'` to emit PHP warnings when a retry is issued.
`client.backoff.logger.template`	Optional template to use for exponential backoff log messages. See the `Guzzle\Plugin\Backoff\BackoffLogger` class for formatting information.

Specifying a region

Some clients require a `region` configuration setting. You can find out if the client you are using requires a region and the regions available to a client by consulting the documentation for that particular client (see *Service-Specific Guides*).

Here's an example of creating an Amazon DynamoDB client that uses the `us-west-1` region:

```
require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Create a client that uses the us-west-1 region
$client = DynamoDbClient::factory(array(
    'key'    => 'YOUR_AWS_ACCESS_KEY_ID',
    'secret' => 'YOUR_AWS_SECRET_ACCESS_KEY',
    'region' => 'us-west-1'
));
```

Setting a custom endpoint

You can specify a completely customized endpoint for a client using the client's `base_url` option. If the client you are using requires a region, then must still specify the name of the region using the `region` option. Setting a custom endpoint can be useful if you're using a mock web server that emulates a web service, you're testing against a private beta endpoint, or you are trying to a use a new region not yet supported by the SDK.

Here's an example of creating an Amazon DynamoDB client that uses a completely customized endpoint:

```
require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

// Create a client that that contacts a completely customized base URL
```

```
$client = DynamoDbClient::factory(array(
    'base_url' => 'http://my-custom-url',
    'region'   => 'my-region-1',
    'key'      => 'abc',
    'secret'   => '123'
));
```

If your custom endpoint uses signature version 4 and must be signed with custom signature scoping values, then you can specify the signature scoping values using `signature.service` (the scoped name of the service) and `signature.region` (the region that you are contacting). These values are typically not required.

Using a proxy

You can send requests with the AWS SDK for PHP through a proxy using the "request options" of a client. These "request options" are applied to each HTTP request sent from the client. One of the option settings that can be specified is the *proxy* option.

Request options are passed to a client through the client's factory method:

```
use Aws\S3\S3Client;

$s3 = S3Client::factory(array(
    'request.options' => array(
        'proxy' => '127.0.0.1:123'
    )
));
```

The above example tells the client that all requests should be proxied through an HTTP proxy located at the *127.0.0.1* IP address using port *123*.

You can supply a username and password when specifying your proxy setting if needed, using the format of `username:password@host:port`.

Command Objects

Command objects are fundamental to how the SDK works. In normal usage of the SDK, you may never interact with command objects. However, if you are *performing operations in parallel*, *inspecting data from the request or response*, or writing custom plugins, you will need to understand how they work.

Typical SDK usage

You can perform a service **operation** by calling the method of the same name on the client object. For example, to perform the Amazon DynamoDB DescribeTable operation, you must call the `Aws\DynamoDb\DynamoDbClient::describeTable()` method. Operation methods, like `describeTable()`, all accept a single argument that is an associative array of values representing the parameters to the operation. The structure of this array is defined for each operation in the SDK's API Documentation (e.g., see the API docs for describeTable()).

```
$result = $dynamoDbClient->describeTable(array(
    'TableName' => 'YourTableName',
));
```

A peek under the hood

If you examine a client class, you will see that the methods corresponding to the operations do not actually exist. They are implemented using the `__call()` magic method behavior. These pseudo-methods are actually shortcuts that encapsulate the SDK's — and the underlying Guzzle library's — use of command objects.

For example, you could perform the same `DescribeTable` operation from the preceding section using command objects:

```
$command = $dynamoDbClient->getCommand('DescribeTable', array(
    'TableName' => 'YourTableName',
));
$result = $command->getResult();
```

A **Command** is an object that represents the execution of a service operation. Command objects are an abstraction of the process of formatting a request to a service, executing the request, receiving the response, and formatting the results. Commands are created and executed by the client and contain references to **Request** and **Response** objects. The **Result** object is a what we refer to as a *"modeled response"*.

Using command objects

Using the pseudo-methods for performing operations is shorter and preferred for typical use cases, but command objects provide greater flexibility and access to additional data.

Manipulating command objects before execution

When you create a command using a client's `getCommand()` method, it does not immediately execute. Because commands are lazily executed, it is possible to pass the command object around and add or modify the parameters. The following examples show how to work with command objects:

```
// You can add parameters after instantiation
$command = $s3Client->getCommand('ListObjects');
$command->set('MaxKeys', 50);
$command->set('Prefix', 'foo/baz/');
$result = $command->getResult();

// You can also modify parameters
$command = $s3Client->getCommand('ListObjects', array(
    'MaxKeys' => 50,
    'Prefix'  => 'foo/baz/',
));
$command->set('MaxKeys', 100);
$result = $command->getResult();

// The set method is chainable
$result = $s3Client->getCommand('ListObjects')
    ->set('MaxKeys', 50);
    ->set('Prefix', 'foo/baz/');
    ->getResult();

// You can also use array access
$command = $s3Client->getCommand('ListObjects');
$command['MaxKeys'] = 50;
$command['Prefix'] = 'foo/baz/';
$result = $command->getResult();
```

Also, see the API docs for commands.

Request and response objects

From the command object, you can access the request, response, and result objects. The availability of these objects depend on the state of the command object.

Managing command state

Commands must be prepared before the request object is available, and commands must executed before the response and result objects are available.

```
// 1. Create
$command = $client->getCommand('OperationName');
```

```
// 2. Prepare
$command->prepare();
$request = $command->getRequest();
// Note: `prepare()` also returns the request object

// 3. Execute
$command->execute();
$response = $command->getResponse();
$result = $command->getResult();
// Note: `execute()` also returns the result object
```

This is nice, because it gives you a chance to modify the request before it is actually sent.

```
$command = $client->getCommand('OperationName');
$request = $command->prepare();
$request->addHeader('foo', 'bar');
$result = $command->execute();
```

You don't have to manage each aspect of the state though, calling `execute()` will also prepare the command, and calling `getResult()` will prepare and execute the command.

Using requests and responses

Request and response objects contain data about the actual requests and responses to the service.

```
$command = $client->getCommand('OperationName');
$command->execute();

// Get and use the request object
$request = $command->getRequest();
$contentLength = $request->getHeader('Content-Length');
$url = $request->getUrl();

// Get and use the response object
$response = $command->getResponse();
$success = $response->isSuccessful();
$status = $response->getStatusCode();
```

You can also take advantage of the `__toString` behavior of the request and response objects. If you print them (e.g., `echo $request;`), you can see the raw request and response data that was sent over the wire.

To learn more, read the API docs for the Request and Response classes.

Executing commands in parallel

The AWS SDK for PHP allows you to execute multiple operations in parallel when you use command objects. This can reduce the total time (sometimes drastically) it takes to perform a set of operations, since you can do them at the same time instead of one after another. The following shows an example of how you could upload two files to Amazon S3 at the same time.

```
$commands = array();
$commands[] = $s3Client->getCommand('PutObject', array(
    'Bucket' => 'SOME_BUCKET',
    'Key'    => 'photos/photo01.jpg',
    'Body'   => fopen('/tmp/photo01.jpg', 'r'),
));
$commands[] = $s3Client->getCommand('PutObject', array(
    'Bucket' => 'SOME_BUCKET',
    'Key'    => 'photos/photo02.jpg',
    'Body'   => fopen('/tmp/photo02.jpg', 'r'),
));
```

```
// Execute an array of command objects to do them in parallel
$s3Client->execute($commands);

// Loop over the commands, which have now all been executed
foreach ($commands as $command) {
    $result = $command->getResult();
    // Do something with result
}
```

Error handling with parallel commands

When executing commands in parallel, error handling becomes a bit trickier. If an exception is thrown, then the SDK (via Guzzle) will aggregate the exceptions together and throw a single `Guzzle\Service\Exception\CommandTransferException` (see the API docs) once all of the commands have completed execution. This exception class keeps track of which commands succeeded and which failed and also allows you to fetch the original exceptions thrown for failed commands.

```
use Guzzle\Service\Exception\CommandTransferException;

try {
    $succeeded = $client->execute($commands);
} catch (CommandTransferException $e) {
    $succeeded = $e->getSuccessfulCommands();
    echo "Failed Commands:\n";
    foreach ($e->getFailedCommands() as $failedCommand) {
        echo $e->getExceptionForFailedCommand($failedCommand)->getMessage() . "\n";
    }
}
```

Waiters

Introduction

One of the higher-level abstractions provided by the SDK are **waiters**. Waiters help make it easier to work with *eventually consistent* systems by providing an easy way to wait until a resource enters into a particular state by polling the resource. You can find a list of the waiters supported by a client by viewing the API Documentation of a service client. Any method with a name starting with "`waitUntil`" will create and invoke a waiter.

In the following example, the Amazon S3 Client is used to create a bucket. Then the waiter method is used to wait until the bucket exists.

```
// Create a bucket
$s3Client->createBucket(array('Bucket' => 'my-bucket'));

// Wait until the created bucket is available
$s3Client->waitUntilBucketExists(array('Bucket' => 'my-bucket'));
```

If the waiter has to poll the bucket too many times, it will throw an `Aws\Common\Exception\RuntimeException` exception.

The "`waitUntil[…]`" methods are all implemented via the `__call` magic method, and are a more discoverable shortcut to using the concrete `waitUntil()` method, since many IDEs can auto-complete methods defined using the `@method` annotation. The following code uses the `waitUntil()` method, but is equivalent to the previous code sample.

```
$s3Client->waitUntil('BucketExists', array('Bucket' => 'my-bucket'));
```

Basic Configuration

You can tune the number of polling attempts issued by a waiter or the number of seconds to delay between each poll by passing optional values prefixed with "waiter.":

```
$s3Client->waitUntilBucketExists(array(
    'Bucket'              => 'my-bucket',
    'waiter.interval'     => 10,
    'waiter.max_attempts' => 3
));
```

Waiter Objects

To interact with the waiter object directly, you must use the `getWaiter()` method. The following code is equivalent to the example in the preceding section.

```
$bucketExistsWaiter = $s3Client->getWaiter('BucketExists')
    ->setConfig(array('Bucket' => 'my-bucket'))
    ->setInterval(10)
    ->setMaxAttempts(3);
$bucketExistsWaiter->wait();
```

Waiter Events

One benefit of working directly with the waiter object is that you can attach event listeners. Waiters emit up to two events in each **wait cycle**. A wait cycle does the following:

1. Dispatch the `waiter.before_attempt` event.
2. Attempt to resolve the wait condition by making a request to the service and checking the result.
3. If the wait condition is resolved, the wait cycle exits. If `max_attempts` is reached, an exception is thrown.
4. Dispatch the `waiter.before_wait` event.
5. Sleep `interval` amount of seconds.

Waiter objects extend the `Guzzle\Common\AbstractHasDispatcher` class which exposes the `addSubscriber()` method and `getEventDispatcher()` method. To attach listeners, you can use the following example, which is a modified version of the previous one.

```
// Get and configure the waiter object
$waiter = $s3Client->getWaiter('BucketExists')
    ->setConfig(array('Bucket' => 'my-bucket'))
    ->setInterval(10)
    ->setMaxAttempts(3);

// Get the event dispatcher and register listeners for both events emitted by the waiter
$dispatcher = $waiter->getEventDispatcher();
$dispatcher->addListener('waiter.before_attempt', function () {
    echo "Checking if the wait condition has been met…\n";
});
$dispatcher->addListener('waiter.before_wait', function () use ($waiter) {
    $interval = $waiter->getInterval();
    echo "Sleeping for {$interval} seconds…\n";
});

$waiter->wait();
```

Custom Waiters

It is possible to implement custom waiter objects if your use case requires application-specific waiter logic or waiters that are not yet supported by the SDK. You can use the `getWaiterFactory()` and `setWaiterFactory()` methods on the client to manipulate the waiter factory used by the client such that your custom waiter can be instantiated. By default the service clients use a `Aws\Common\Waiter\CompositeWaiterFactory` which allows you to add additional factories if needed. The following example shows how to implement a contrived custom waiter class and then modify a client's waiter factory such that it can create instances of the custom waiter.

```
namespace MyApp\FakeWaiters
{
    use Aws\Common\Waiter\AbstractResourceWaiter;

    class SleptThreeTimes extends AbstractResourceWaiter
    {
        public function doWait()
        {
            if ($this->attempts < 3) {
                echo "Need to sleep…\n";
                return false;
            } else {
                echo "Now I've slept 3 times.\n";
                return true;
            }
        }
    }
}

namespace
{
    use Aws\S3\S3Client;
    use Aws\Common\Waiter\WaiterClassFactory;

    $s3Client = S3Client::factory();

    $compositeFactory = $s3Client->getWaiterFactory();
    $compositeFactory->addFactory(new WaiterClassFactory('MyApp\FakeWaiters'));

    $waiter = $s3Client->waitUntilSleptThreeTimes();
}
```

The result of this code should look like the following:

```
Need to sleep…
Need to sleep…
Need to sleep…
Now I've slept 3 times.
```

Waiter Definitions

The waiters that are included in the SDK are defined in the service description for their client. They are defined using a configuration DSL (domain-specific language) that describes the default wait intervals, wait conditions, and how to check or poll the resource to resolve the condition.

This data is automatically consumed and used by the `Aws\Common\Waiter\WaiterConfigFactory` class when a client is instantiated so that the waiters defined in the service description are available to the client.

The following is an excerpt of the Amazon Glacier service description that defines the waiters provided by `Aws\Glacier\GlacierClient`.

```
return array(
    // ...

    'waiters' => array(
        '__default__' => array(
            'interval' => 3,
            'max_attempts' => 15,
        ),
        '__VaultState' => array(
            'operation' => 'DescribeVault',
        ),
```

```
        'VaultExists' => array(
            'extends' => '__VaultState',
            'success.type' => 'output',
            'description' => 'Wait until a vault can be accessed.',
            'ignore_errors' => array(
                'ResourceNotFoundException',
            ),
        ),
        'VaultNotExists' => array(
            'extends' => '__VaultState',
            'description' => 'Wait until a vault is deleted.',
            'success.type' => 'error',
            'success.value' => 'ResourceNotFoundException',
        ),
    ),

    // ...
);
```

In order for you to contribute waiters to the SDK, you will need to implement them using the waiters DSL. The DSL is not documented yet, since it is currently subject to change, so if you are interested in helping to implement more waiters, please reach out to us via GitHub.

Iterators

Introduction

Some AWS operations return truncated results that require subsequent requests in order to retrieve the entire result set. The subsequent requests typically require pagination tokens or markers from the previous request in order to retrieve the next set of results. Working with these tokens can be cumbersome, since you must manually keep track of them, and the API for each service may differ in how it uses them.

The AWS SDK for PHP has a feature called **iterators** that allow you to retrieve an *entire* result set without manually handling pagination tokens or markers. The iterators in the SDK implement PHP's `Iterator` interface, which allows you to easily enumerate or iterate through resources from a result set with `foreach`.

You can find a list of the iterators supported by a client by viewing the docblock of a client. Any `@method` tag that has a name that looks like "`get[…]Iterator`" will return an iterator. For example, the following code uses the `getListObjectsIterator()` method of the S3 client object to create an iterator for objects in a bucket.

```
$iterator = $client->getListObjectsIterator(array('Bucket' => 'my-bucket'));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

The "`get[…]Iterator`" methods are all implemented via the `__call` magic method, and are a more discoverable shortcut to using the concrete `getIterator()` method, since many IDEs can auto-complete methods defined using the `@method` annotation. The following code uses the `getIterator()` method, but is equivalent to the previous code sample.

```
$iterator = $client->getIterator('ListObjects', array('Bucket' => 'my-bucket'));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

The `getIterator()` method also accepts a command object for the first argument. If you have a command object already instantiated, you can create an iterator directly from the command object.

```
$command = $client->getCommand('ListObjects', array('Bucket' => 'my-bucket'));
$iterator = $client->getIterator($command);
```

Iterator Objects

The actual object returned by `getIterator()`, and any `get[…]Iterator()` method, is an instance of the `Aws\Common\Iterator\AwsResourceIterator` class (see the API docs for more information about its methods and properties). This class implements PHP's native `Iterator` interface, which is why it works with `foreach`, can be used with iterator functions like `iterator_to_array`, and integrates well with SPL iterators like `LimitIterator`.

Iterator objects only store one "page" of results at a time and only make as many requests as they need based on the current iteration. The S3 `ListObjects` operation only returns up to 1000 objects at a time. If your bucket has ~10000 objects, then the iterator would need to do 10 requests. However, it does not execute the subsequent requests until needed. If you are iterating through the results, the first request would happen when you start iterating, and the second request would not happen until you iterate to the 1001th object. This can help your application save memory by only holding one page of results at a time.

Basic Configuration

Iterators accept an extra set of parameters that are not passed into the commands. You can set a limit on the number of results you want with the `limit` parameter, and you can control how many results you want to get back per request using the `page_size` parameter. If no `limit` is specified, then all results are retrieved. If no `page_size` is specified, then the iterator will use the maximum page size allowed by the operation being executed.

The following example will make 10 Amazon S3 `ListObjects` requests (assuming there are more than 1000 objects in the specified bucket) that each return up to 100 objects. The `foreach` loop will yield up to 999 objects.

```
$iterator = $client->getListObjectsIterator(array(
    'Bucket' => 'my-bucket'
), array(
    'limit'     => 999,
    'page_size' => 100
));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

There are some limitations to the `limit` and `page_size` parameters though. Not all operations support specifying a page size or limit, so the iterator will do its best with what you provide. For example, if an operation always returns 1000 results, and you specify a limit of 100, the iterator will only yield 100 results, even though the actual request sent to the service yielded 1000.

Iterator Events

Iterators emit 2 kinds of events:

1. `resource_iterator.before_send` - Emitted right before a request is sent to retrieve results.
2. `resource_iterator.after_send` - Emitted right after a request is sent to retrieve results.

Iterator objects extend the `Guzzle\Common\AbstractHasDispatcher` class which exposes the `addSubscriber()` method and the `getEventDispatcher()` method. To attach listeners, you can use the following example which echoes a message right before and after a request is executed by the iterator.

```
$iterator = $client->getListObjectsIterator(array(
    'Bucket' => 'my-bucket'
));

// Get the event dispatcher and register listeners for both events
$dispatcher = $iterator->getEventDispatcher();
$dispatcher->addListener('resource_iterator.before_send', function ($event) {
    echo "Getting more results…\n";
});
$dispatcher->addListener('resource_iterator.after_send', function ($event) use ($iterator) {
    $requestCount = $iterator->getRequestCount();
```

```
    echo "Results received. {$requestCount} request(s) made so far.\n";
});

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

Modeled Responses

Introduction

The result of a performing an operation is what we refer to as a **modeled response**. Instead of returning the raw XML or JSON data, the SDK will coerce the data into an associative array and normalize some aspects of the data based on its knowledge of the specific service and the underlying response structure.

The actual value returned is a Model (`Guzzle\Service\Resource\Model`) object. The Model class is a part of the SDK's underlying Guzzle library, but you do not need to know anything about Guzzle to use your operation results. The Model object contains the data from the response and can be used like an array (e.g., `$result['Table']`). It also has convenience methods like `get()`, `getPath()`, and `toArray()`. The contents of the modeled response depend on the operation that was executed and are documented in the API docs for each operation (e.g., see the *Returns* section in the API docs for the DynamoDB DescribeTable operation).

```
$result = $dynamoDbClient->describeTable(array(
    'TableName' => 'YourTableName',
));

// Get a specific value from the result
$table = $result['Table'];
if ($table && isset($table['TableStatus'])) {
    echo $table['TableStatus'];
}
//> ACTIVE

// Get nested values from the result easily
echo $result->getPath('Table/TableStatus');
//> ACTIVE

// Convert the Model to a plain array
var_export($result->toArray());
//> array ( 'Table' => array ( 'AttributeDefinitions' => array ( ... ) ... ) ... )
```

Working with Model objects

Model objects (and Command objects) inherit from the Guzzle Collection class and implement PHP's native `ArrayAccess`, `IteratorAggregate`, and `Countable` interfaces. This means that they behave like arrays when you are accessing keys and iterating over key-value pairs. You can also use the `toArray()` method of the Model object to get the array form directly.

However, model objects will not throw errors on undefined keys, so it's safe to use values directly without doing `isset()` checks. It the key doesn't exist, then the value will be returned as `null`.

```
// Use an instance of S3Client to get an object
$result = $s3Client->getObject(array(
    'Bucket' => 'my-bucket',
    'Key'    => 'test.txt'
));

// Using a value that may not exist
if (!$result['ContentLength']) {
    echo "Empty file.";
```

```
}

$isDeleted = (bool) $result->get('DeleteMarker');
```

Of course, you can still use `isset()` checks if you want to, since `Model` does implement `ArrayAccess`. The model object (and underlying Collection object) also has convenience methods for finding and checking for keys and values.

```
// You can use isset() since the object implements ArrayAccess
if (!isset($result['ContentLength'])) {
    echo "Empty file.";
}

// There is also a method that does the same type of check
if (!$result->hasKey('ContentLength')) {
    echo "Empty file.";
}

// If needed, you can search for a key in a case-insensitive manner
echo $result->keySearch('body');
//> Body
echo $result->keySearch('Body');
//> Body

// You can also list all of the keys in the result
var_export($result->getKeys());
//> array ( 'Body', 'DeleteMarker', 'Expiration', 'ContentLength', ... )

// The getAll() method will return the result data as an array
// You can specify a set of keys to only get a subset of the data
var_export($result->getAll(array('Body', 'ContentLength')));
//> array ( 'Body' => 'Hello!' , 'ContentLength' => 6 )
```

Getting nested values

The `getPath()` method of the model is useful for easily getting nested values from a response. The path is specified as a series of keys separated by slashes.

```
// Perform a RunInstances operation and traverse into the results to get the InstanceId
$result = $ec2Client->runInstances(array(
    'ImageId'      => 'ami-548f13d',
    'MinCount'     => 1,
    'MaxCount'     => 1,
    'InstanceType' => 't1.micro',
));
$instanceId = $result->getPath('Instances/0/InstanceId');
```

Wildcards are also supported so that you can get extract an array of data. The following example is a modification of the preceding such that multiple InstanceIds can be retrieved.

```
// Perform a RunInstances operation and get an array of the InstanceIds that were created
$result = $ec2Client->runInstances(array(
    'ImageId'      => 'ami-548f13d',
    'MinCount'     => 3,
    'MaxCount'     => 5,
    'InstanceType' => 't1.micro',
));
$instanceId = $result->getPath('Instances/*/InstanceId');
```

Using data in the model

Response Models contain the parsed data from the response from a service operation, so the contents of the model will be different depending on which operation you've performed.

The SDK's API docs are the best resource for discovering what the model object will contain for a given operation. The API docs contain a full specification of the data in the response model under the *Returns* section of the docs for an operation (e.g., S3 GetObject operation, EC2 RunInstances operation).

From within your code you can convert the response model directly into an array using the `toArray()` method. If you are doing some debugging in your code, you could use `toArray()` in conjunction with `print_r()` to print out a simple representation of the response data.

```
$result = $ec2Client->runInstances(array(/* ... */));
print_r($result->toArray());
```

You can also examine the service description for a service, which is located in the `Resources` directory within a given client's namespace directory. For example, here is a snippet from the SQS service description (located in `src/Aws/Sqs/Resources/`) that shows the schema for the response of the `SendMessage` operation.

```
// ...
    'SendMessageResult' => array(
        'type' => 'object',
        'additionalProperties' => true,
        'properties' => array(
            'MD5OfMessageBody' => array(
                'description' => 'An MD5 digest of the non-URL-encoded message body string.
                'type' => 'string',
                'location' => 'xml',
            ),
            'MessageId' => array(
                'description' => 'The message ID of the message added to the queue.',
                'type' => 'string',
                'location' => 'xml',
            ),
        ),
    ),
// ...
```

Getting Response Headers

The `Response` object is not directly accessible from the `Model` object. If you are interested in getting header values, the status code, or other data from the response you will need to get the `Response` object from the `Command` object (see *Command Objects*). You may need to switch from using the shorthand command syntax to the expanded syntax so that the command object can be accessed directly.

```
// Getting the response Model with the shorthand syntax
$result = $s3Client->createBucket(array(/* ... */));

// Getting the response Model with the expanded syntax
$command = $s3Client->getCommand('CreateBucket', array(/* ... */));
$result = $command->getResult();

// Getting the Response object from the Command
$response = $command->getResponse();
$contentLength = $response->getHeader('Content-Length');
$statusCode = $response->getStatusCode();
```

In some cases, particularly with REST-like services like Amazon S3 and Amazon Glacier, most of the important headers are already included in the response model.

Static Client Facades

Introduction

Version 2.4 of the AWS SDK for PHP adds the ability to enable and use static client "facades". These facades provide an easy, static interface to service clients available in the service builder. For example, when working with a normal client instance, you might have code that looks like the following:

```
// Get the configured S3 client from the service builder
$s3 = $aws->get('s3');

// Execute the CreateBucket command using the S3 client
$s3->createBucket(array('Bucket' => 'your_new_bucket_name'));
```

With client facades enabled, this can also be accomplished with the following code:

```
// Execute the CreateBucket command using the S3 client
S3::createBucket(array('Bucket' => 'your_new_bucket_name'));
```

Why Use Client Facades?

The use of static client facades is completely optional. We have included this feature in the SDK in order to appeal to PHP developers who prefer static notation or who are familiar with PHP frameworks like Code Ignitor, Laravel, or Kohana where this style of method invocation is common.

Though using static client facades has little real benefit over using client instances, it can make your code more concise and prevent your from having to inject the service builder or client instance into the context of where you need the client object. This can make your code easier to write and understand. Whether or not you should use the client facades is purely a matter of preference.

The way in which client facades work in the AWS SDK for PHP is similar to how facades work in the Laravel 4 Framework. Even though you are calling static classes, all of the method calls are proxied to method calls on actual client instances — the ones stored in the service builder. This means that the usage of the clients via the client facades can still be mocked in your unit tests, which removes one of the general disadvantages to using static classes in object-oriented programming. For information about how to test code that uses client facades, please see the **Testing Code that Uses Client Facades** below.

Enabling and Using Client Facades

To enable static client facades to be used in your application, you must use the `Aws\Common\Aws::enableFacades` method when you setup the service builder.

```
// Include the Composer autoloader
require 'vendor/autoload.php';

// Instantiate the SDK service builder with my config and enable facades
$aws = Aws::factory('/path/to/my_config.php')->enableFacades();
```

This will setup the client facades and alias them into the global namespace. After that, you can use them anywhere to have more simple and expressive code for interacting with AWS services.

```
// List current buckets
echo "Current Buckets:\n";
foreach (S3::getListBucketsIterator() as $bucket) {
    echo "{$bucket['Name']}\n";
}

$args = array('Bucket' => 'your_new_bucket_name');
$file = '/path/to/the/file/to/upload.jpg';

// Create a new bucket and wait until it is available for uploads
S3::createBucket($args) and S3::waitUntilBucketExists($args);
echo "\nCreated a new bucket: {$args['Bucket']}.\n";

// Upload a file to the new bucket
$result = S3::putObject($args + array(
    'Key'  => basename($file),
```

```
    'Body' => fopen($file, 'r'),
));
echo "\nCreated a new object: {$result['ObjectURL']}\n";
```

You can also mount the facades into a namespace other than the global namespace. For example, if you wanted to make the client facades available in the "Services" namespace, then you could do the following:

```
Aws::factory('/path/to/my_config.php')->enableFacades('Services');

$result = Services\DynamoDb::listTables();
```

The client facades that are available are determined by what is in your service builder configuration (see *Configuring the SDK*). If you are extending the SDK's default configuration file or not providing one at all, then all of the clients should be accessible from the service builder instance and client facades (once enabled) by default.

Based on the following excerpt from the default configuration file (located at `src/Aws/Common/Resources/aws-config.php`):

```
's3' => array(
    'alias'   => 'S3',
    'extends' => 'default_settings',
    'class'   => 'Aws\S3\S3Client'
),
```

The `'class'` key indicates the client class that the static client facade will proxy to, and the `'alias'` key indicates what the client facade will be named. Only entries in the service builder config that have both the `'alias'` and `'class'` keys specified will be mounted as static client facades. You can potentially update or add to your service builder config to alter or create new or custom client facades.

Testing Code that Uses Client Facades

With the static client facades in the SDK, even though you are calling static classes, all of the method calls are proxied to method calls on actual client instances — the ones stored in the service builder. This means that they can be mocked during tests, which removes one of the general disadvantages to using static classes in object-oriented programming.

To mock a client facade for a test, you can explicitly set a mocked client object for the key in the service builder that would normally contain the client referenced by the client facade. Here is a complete, but contrived, PHPUnit test showing how this is done:

```
<?php

use Aws\Common\Aws;
use Guzzle\Service\Resource\Model;
use YourApp\Things\FileBrowser;

class SomeKindOfFileBrowserTest extends PHPUnit_Framework_TestCase
{
    private $serviceBuilder;

    public function setUp()
    {
        $this->serviceBuilder = Aws::factory();
        $this->serviceBuilder->enableFacades();
    }

    public function testCanDoSomethingWithYourAppsFileBrowserClass()
    {
        // Mock the ListBuckets method of S3 client
        $mockS3Client = $this->getMockBuilder('Aws\S3\S3Client')
            ->disableOriginalConstructor()
            ->getMock();
        $mockS3Client->expects($this->any())
```

```
            ->method('listBuckets')
            ->will($this->returnValue(new Model(array(
                'Buckets' => array(
                    array('Name' => 'foo'),
                    array('Name' => 'bar'),
                    array('Name' => 'baz')
                )
            ))));
        $this->serviceBuilder->set('s3', $mockS3Client);

        // Test the FileBrowser object that uses the S3 client facade internally
        $fileBrowser = new FileBrowser();
        $partitions = $fileBrowser->getPartitions();
        $this->assertEquals(array('foo', 'bar', 'baz'), $partitions);
    }
}
```

Alternatively, if you are specifically only mocking responses from clients, you might consider using the Guzzle Mock Plugin.

Performance Guide

The AWS SDK for PHP is able to send HTTP requests to various web services with minimal overhead. This document serves as a guide that will help you to achieve optimal performance with the SDK.

Upgrade PHP 42

Use PHP 5.5 or an opcode cache like APC 42

Use Composer with a classmap autoloader 44

Uninstall Xdebug 44

Install PECL uri_template 44

Turn off parameter validation 44

Cache instance profile credentials 45

Check if you are being throttled 45

Preload frequently included files 45

Profile your code to find performance bottlenecks 46

Comparing SDK1 and SDK2 46

Upgrade PHP

Using an up-to-date version of PHP will generally improve the performance of your PHP applications. Did you know that PHP 5.4 is 20-40% faster than PHP 5.3? Upgrading to PHP 5.4 or greater will provide better performance and lower memory usage. If you cannot upgrade from PHP 5.3 to PHP 5.4 or PHP 5.5, upgrading to PHP 5.3.18 or greater will improve performance over older versions of PHP 5.3.

You can install PHP 5.4 on an Amazon Linux AMI using the following command.

```
yum install php54
```

Use PHP 5.5 or an opcode cache like APC

To improve the overall performance of your PHP environment, it is highly recommended that you use an opcode cache such as the OPCache built into PHP 5.5, APC, XCache, or WinCache. By default, PHP must load a file from disk, parse the PHP code into opcodes, and finally execute the opcodes. Installing an opcode cache allows the parsed opcodes to be cached in memory so that you do not need to parse the script on every web server request, and in ideal circumstances, these opcodes can be served directly from memory.

We have taken great care to ensure that the SDK will perform well in an environment that utilizes an opcode cache.

> ***Note***
>
> PHP 5.5 comes with an opcode cache that is installed and enabled by default: http://php.net/manual/en/book.opcache.php
>
> If you are using PHP 5.5, then you may skip the remainder of this section.

APC

If you are not able to run PHP 5.5, then we recommend using APC as an opcode cache.

Installing on Amazon Linux

When using Amazon Linux, you can install APC using one of the following commands depending on if you are using PHP 5.3 or PHP 5.4.

```
# For PHP 5.4
yum install php54-pecl-apc

# For PHP 5.3
yum install php-pecl-apc
```

Modifying APC settings

APC configuration settings can be set and configured in the `apc.ini` file of most systems. You can find more information about configuring APC in the PHP.net APC documentation.

The APC configuration file is located at `/etc/php.d/apc.ini` on Amazon Linux.

```
# You can only modify the file as sudo
sudo vim /etc/php.d/apc.ini
```

apc.shm_size=128M

It is recommended that you set the apc.shm_size setting to be 128M or higher. You should investigate what the right value will be for your application. The ideal value will depend on how many files your application includes, what other frameworks are used by your application, and if you are caching data in the APC user cache.

You can run the following command on Amazon Linux to set apc.shm_size to 128M:

```
sed -i "s/apc.shm_size=.*/apc.shm_size=128M/g" /etc/php.d/apc.ini
```

apc.stat=0

The SDK adheres to PSR-0 and relies heavily on class autoloading. When `apc.stat=1`, APC will perform a stat on each cached entry to ensure that the file has not been updated since it was cache in APC. This incurs a system call for every autoloaded class required by a PHP script (you can see this for yourself by running `strace` on your application).

You can tell APC to not stat each cached file by setting `apc.stat=0` in you apc.ini file. This change will generally improve the overall performance of APC, but it will require you to explicitly clear the APC cache when a cached file should be updated. This can be accomplished with Apache by issuing a hard or graceful restart. This restart step could be added as part of the deployment process of your application.

You can run the following command on Amazon Linux to set apc.stat to 0:

```
sed -i "s/apc.stat=1/apc.stat=0/g" /etc/php.d/apc.ini
```

From the PHP documentation

This defaults to on, forcing APC to stat (check) the script on each request to determine if it has been modified. If it has been modified it will recompile and cache the new version. If this setting is off, APC will not check, which usually means that to force APC to recheck files, the web server will have to be restarted or the cache will have to be manually cleared. Note that FastCGI web server configurations may not clear the cache on restart. On a production server where the script files rarely change, a significant performance boost can be achieved by disabled stats.

For included/required files this option applies as well, but note that for relative path includes (any path that doesn't start with / on Unix) APC has to check in order to uniquely identify the file. If you use absolute path includes APC can skip the stat and use that absolute path as the unique identifier for the file.

Use Composer with a classmap autoloader

Using Composer is the recommended way to install the AWS SDK for PHP. Composer is a dependency manager for PHP that can be used to pull in all of the dependencies of the SDK and generate an autoloader.

Autoloaders are used to lazily load classes as they are required by a PHP script. Composer will generate an autoloader that is able to autoload the PHP scripts of your application and all of the PHP scripts of the vendors required by your application (i.e. the AWS SDK for PHP). When running in production, it is highly recommended that you use a classmap autoloader to improve the autoloader's speed. You can generate a classmap autoloader by passing the `-o` or `--optimize-autoloader` option to Composer's install command:

```
php composer.phar install --optimize-autoloader
```

Please consult the *Installation* guide for more information on how to install the SDK using Composer.

Uninstall Xdebug

Xdebug is an amazing tool that can be used to identify performance bottlenecks. However, if performance is critical to your application, do not install the Xdebug extension on your production environment. Simply loading the extension will greatly slow down the SDK.

When running on Amazon Linux, Xdebug can be removed with the following command:

```
# PHP 5.4
yum remove php54-pecl-xdebug

# PHP 5.3
yum remove php-pecl-xdebug
```

Install PECL uri_template

The SDK utilizes URI templates to power each operation. In order to be compatible out of the box with the majority of PHP environments, the default URI template expansion implementation is written in PHP. PECL URI_Template is a URI template extension for PHP written in C. This C implementation is about 3 times faster than the default PHP implementation for expanding URI templates. Your application will automatically begin utilizing the PECL uri_template extension after it is installed.

```
pecl install uri_template-alpha
```

Turn off parameter validation

The SDK utilizes service descriptions to tell the client how to serialize an HTTP request and parse an HTTP response into a Model object. Along with serialization information, service descriptions are used to validate operation inputs client-side before sending a request. Disabling parameter validation is a micro-optimization, but this setting can typically be disabled in production by setting the `validation` option in a client factory method to `false`.

```
$client = Aws\DynamoDb\DynamoDbClient::factory(array(
    'region'     => 'us-west-2',
```

```
    'validation' => false
));
```

Cache instance profile credentials

When you do not provide credentials to the SDK and do not have credentials defined in your environment variables, the SDK will attempt to utilize IAM instance profile credentials by contacting the Amazon EC2 instance metadata service (IMDS). Contacting the IMDS requires an HTTP request to retrieve credentials from the IMDS.

You can cache these instance profile credentials in memory until they expire and avoid the cost of sending an HTTP request to the IMDS each time the SDK is utilized. Set the `credentials.cache` option to `true` to attempt to utilize the Doctrine Cache PHP library to cache credentials with APC.

```
$client = Aws\DynamoDb\DynamoDbClient::factory(array(
    'region'            => 'us-west-2',
    'credentials.cache' => true
));
```

Note

You will need to install Doctrine Cache in order for the SDK to cache credentials when setting `credentials.cache` to `true`. You can add doctrine/cache to your composer.json dependencies by adding to your project's `required` section:

```
{
    "required": {
        "aws/sdk": "2.*",
        "doctrine/cache": "1.*"
    }
}
```

Check if you are being throttled

You can check to see if you are being throttled by enabling the exponential backoff logger option. You can set the `client.backoff.logger` option to `debug` when in development, but we recommend that you provide a `Guzzle\Log\LogAdapterInterface` object when running in production.

```
$client = Aws\DynamoDb\DynamoDbClient::factory(array(
    'region'                => 'us-west-2',
    'client.backoff.logger' => 'debug'
));
```

When using Amazon DynamoDB, you can monitor your tables for throttling using Amazon CloudWatch.

Preload frequently included files

The AWS SDK for PHP adheres to PSR-0 and heavily utilizes class autoloading. Each class is in a separate file and are included lazily as they are required. Enabling an opcode cache like APC, setting `apc.stat=0`, and utilizing an optimized Composer autoloader will help to mitigate the performance cost of autoloading the classes needed to utilize the SDK. In situations like hosting a webpage where you are loading the same classes over and over, you can shave off a bit more time by compiling all of the autoloaded classes into a single file thereby completely eliminating the cost of autoloading. This technique can not only speed up the use of the SDK for specific use cases (e.g. using the Amazon DynamoDB session handler), but can also speed up other aspects of your application. Even with `apc.stat=0`, preloading classes that you know will be used in your application will be slightly faster than relying on autoloading.

You can easily generate a compiled autoloader file using the ClassPreloader project. View the project's README for information on creating a "preloader" for use with the AWS SDK for PHP.

Profile your code to find performance bottlenecks

You will need to profile your application to determine the bottlenecks. This can be done using Xdebug, XHProf, strace, and various other tools. There are many resources available on the internet to help you track down performance problems with your application. Here are a few that we have found useful:

- http://talks.php.net/show/devconf/0
- http://talks.php.net/show/perf_tunning/16

Comparing SDK1 and SDK2

Software performance is very subjective and depends heavily on factors outside of the control of the SDK. The AWS SDK for PHP is tuned to cover the broadest set of performance sensitive applications using AWS. While there may be a few isolated cases where V1 of the the SDK is as fast or faster than V2, that is not generally true and comes with the loss of extensibility, maintainability, persistent HTTP connections, response parsing, PSR compliance, etc.

Depending on your use case, you will find that a properly configured environment running the AWS SDK for PHP is generally just as fast as SDK1 for sending a single request and more than 350% faster than SDK1 for sending many requests.

Comparing batch requests

A common misconception when comparing the performance of SDK1 and SDK2 is that SDK1 is faster than SDK2 when sending requests using the "batch()" API.

SDK1 is generally *not* faster at sending requests in parallel than SDK2. There may be some cases where SDK1 will appear to more quickly complete the process of sending multiple requests in parallel, but SDK1 does not retry throttled requests when using the `batch()` API. In SDK2, throttled requests are automatically retried in parallel using truncated exponential backoff. Automatically retrying failed requests will help to ensure that your application is successfully completing the requests that you think it is.

You can always disable retries if your use case does not benefit from retrying failed requests. To disable retries, set 'client.backoff' to `false` when creating a client.

```
$client = Aws\DynamoDb\DynamoDbClient::factory(array(
    'region'         => 'us-west-2',
    'client.backoff' => false
));
```

Frequently Asked Questions (FAQ)

What methods are available on a client?

The AWS SDK for PHP utilizes service descriptions and dynamic magic __call() methods to execute API operations. Every magic method supported by a client is documented in the docblock of a client class using `@method` annotations. Several PHP IDEs, including PHPStorm and Zend Studio, are able to autocomplete based on `@method` annotations. You can find a full list of methods available for a web service client in the API documentation of the client or in the user guide for that client.

For example, the Amazon S3 client supports the following operations: *Creating a bucket*

What do I do about a cURL SSL certificate error?

This issue can occur when using an out of date CA bundle with cURL and SSL. You can get around this issue by updating the CA bundle on your server or downloading a more up to date CA bundle from the cURL website directly.

Simply download a more up to date CA bundle somewhere on your system and instruct the SDK to use that CA bundle rather than the default. You can configure the SDK to use a more up to date CA bundle by specifying the `ssl.certificate_authority` in a client's factory method or the configuration settings used with `Aws\Common\Aws`.

```
$aws = Aws\Common\Aws::factory(array(
    'region' => 'us-west-2',
    'key'    => '****',
    'secret' => '****',
    'ssl.certificate_authority' => '/path/to/updated/cacert.pem'
));
```

You can find out more about how cURL bundles the CA bundle here: http://curl.haxx.se/docs/caextract.html

How do I disable SSL?

Warning

Because SSL requires all data to be encrypted and requires more TCP packets to complete a connection handshake than just TCP, disabling SSL may provide a small performance improvement. However, with SSL disabled, all data is sent over the wire unencrypted. Before disabling SSL, you must carefully consider the security implications and the potential for eavesdropping over the network.

You can disable SSL by setting the `scheme` parameter in a client factory method to 'http'.

```
$client = Aws\DynamoDb\DynamoDbClient::factory(array(
    'region' => 'us-west-2',
    'scheme' => 'http'
));
```

How can I make the SDK faster?

See *Performance Guide* for more information.

Why can't I upload or download files greater than 2GB?

Because PHP's integer type is signed and many platforms use 32-bit integers, the AWS SDK for PHP does not correctly handle files larger than 2GB on a 32-bit stack (where "stack" includes CPU, OS, web server, and PHP binary). This is a well-known PHP issue. In the case of Microsoft® Windows®, there are no official builds of PHP that support 64-bit integers.

The recommended solution is to use a 64-bit Linux stack, such as the 64-bit Amazon Linux AMI with the latest version of PHP installed.

For more information, please see: PHP filesize :Return values.

How can I see what data is sent over the wire?

You can attach a `Guzzle\Plugin\Log\LogPlugin` to any client to see all request and response data sent over the wire. The LogPlugin works with any logger that implements the `Guzzle\Log\LogAdapterInterface` interface (currently Monolog, ZF1, ZF2).

If you just want to quickly see what data is being sent over the wire, you can simply attach a debug log plugin to your client.

```
use Guzzle\Plugin\Log\LogPlugin;

// Create an Amazon S3 client
$s3Client = S3Client::factory();

// Add a debug log plugin
$s3Client->addSubscriber(LogPlugin::getDebugPlugin());
```

For more complex logging or logging to a file, you can build a LogPlugin manually.

```
use Guzzle\Common\Log\MonologLogAdapter;
use Guzzle\Plugin\Log\LogPlugin;
use Monolog\Logger;
use Monolog\Handler\StreamHandler;

// Create a log channel
$log = new Logger('aws');
$log->pushHandler(new StreamHandler('/path/to/your.log', Logger::WARNING));

// Create a log adapter for Monolog
$logger = new MonologLogAdapter($log);

// Create the LogPlugin
$logPlugin = new LogPlugin($logger);

// Create an Amazon S3 client
$s3Client = S3Client::factory();

// Add the LogPlugin to the client
$s3Client->addSubscriber($logPlugin);
```

You can find out more about the LogPlugin on the Guzzle website: http://guzzlephp.org/guide/plugins.html#log-plugin

How can I set arbitrary headers on a request?

You can add any arbitrary headers to a service operation by setting the `command.headers` value. The following example shows how to add an `X-Foo-Baz` header to an Amazon S3 PutObject operation.

```
$s3Client = S3Client::factory();
$s3Client->putObject(array(
    'Key'    => 'test',
    'Bucket' => 'mybucket',
    'command.headers' => array(
        'X-Foo-Baz' => 'Bar'
    )
));
```

Does the SDK follow semantic versioning?

Yes. The SDK follows a semantic versioning scheme similar to – but not the same as – semver. Instead of the **MAJOR.MINOR.PATCH** scheme specified by semver, the SDK actually follows a scheme that looks like **PARADIGM.MAJOR.MINOR** where:

1. The **PARADIGM** version number is incremented when **drastic, breaking changes** are made to the SDK, such that the fundamental way of using the SDK is different. You are probably aware that version 1.x and version 2.x of the AWS SDK for PHP are *very* different.
2. The **MAJOR** version number is incremented when **breaking changes** are made to the API. These are usually small changes, and only occur when one of the services makes breaking changes changes to their API. Make sure to check the CHANGELOG and UPGRADING documents when these changes occur.
3. The **MINOR** version number is incremented when any **backwards-compatible** change is made, whether it's a new feature or a bug fix.

The best way to ensure that you are not affected by breaking changes is to set your dependency on the SDK in Composer to stay within a particular **PARADIGM.MAJOR** version. This can be done using the wildcard syntax:

```
{
    "require": {
        "aws/aws-sdk-php": "2.4.*"
    }
}
```

...Or by using the the tilde operator. The following statement is equivalent to *>=2.4.9,<2.5*:

```
{
    "require": {
        "aws/aws-sdk-php": "~2.4.9"
    }
}
```

See the Composer documentation for more information on configuring your dependencies.

The SDK may at some point adopt the semver standard, but this will probably not happen until the next paradigm-type change.

Why am I seeing a "Cannot redeclare class" error?

We have observed this error a few times when using the `aws.phar` from the CLI with APC enabled. This is due to some kind of issue with phars and APC. Luckily there are a few ways to get around this. Please choose the one that makes the most sense for your environment and application.

1. **Disable APC for CLI** - Change the `apc.enable_cli` INI setting to `Off`.
2. **Tell APC not to cache phars** - Change the `apc.filters` INI setting to include `"^phar://"`.
3. **Don't use APC** - PHP 5.5, for example, comes with Zend OpCache built in. This problem has not been observed with Zend OpCache.
4. **Don't use the phar** - You can install the SDK through Composer (recommended) or by using the zip file.

What is an InstanceProfileCredentialsException?

If you are seeing an `Aws\Common\Exception\InstanceProfileCredentialsException` while using the SDK, this means that the SDK was not provided with any credentials.

If you instantiate a client *without* credentials, on the first time that you perform a service operation, the SDK will attempt to find credentials. It first checks in some specific environment variables, then it looks for instance profile credentials, which are only available on configured Amazon EC2 instances. If absolutely no credentials are provided or found, an `Aws\Common\Exception\InstanceProfileCredentialsException` is thrown.

If you are seeing this error and you are intending to use instance profile credentials, then you need to make sure that the Amazon EC2 instance that the SDK is running on is configured with an appropriate IAM role.

If you are seeing this error and you are **not** intending to use instance profile credentials, then you need to make sure that you are properly providing credentials to the SDK.

For more information, see *Providing Credentials to the SDK*.

Auto Scaling

This guide focuses on the AWS SDK for PHP client for Auto Scaling. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\AutoScaling\AutoScalingClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\AutoScaling\AutoScalingClient;

$client = AutoScalingClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Auto Scaling is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('AutoScaling');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Auto Scaling Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AttachInstances	CreateAutoScalingGroup
CreateLaunchConfiguration	CreateOrUpdateTags
DeleteAutoScalingGroup	DeleteLaunchConfiguration
DeleteNotificationConfiguration	DeletePolicy
DeleteScheduledAction	DeleteTags
DescribeAccountLimits	DescribeAdjustmentTypes
DescribeAutoScalingGroups	DescribeAutoScalingInstances
DescribeAutoScalingNotificationTypes	DescribeLaunchConfigurations
DescribeMetricCollectionTypes	DescribeNotificationConfigurations
DescribePolicies	DescribeScalingActivities
DescribeScalingProcessTypes	DescribeScheduledActions

DescribeTags	DescribeTerminationPolicyTypes
DisableMetricsCollection	EnableMetricsCollection
ExecutePolicy	PutNotificationConfiguration
PutScalingPolicy	PutScheduledUpdateGroupAction
ResumeProcesses	SetDesiredCapacity
SetInstanceHealth	SuspendProcesses
TerminateInstanceInAutoScalingGroup	UpdateAutoScalingGroup

AWS CloudFormation

This guide focuses on the AWS SDK for PHP client for AWS CloudFormation. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudFormation\CloudFormationClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\CloudFormation\CloudFormationClient;

$client = CloudFormationClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS CloudFormation is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudFormation');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS CloudFormation Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CancelUpdateStack	CreateStack
DeleteStack	DescribeStackEvents
DescribeStackResource	DescribeStackResources
DescribeStacks	EstimateTemplateCost
GetStackPolicy	GetTemplate
ListStackResources	ListStacks
SetStackPolicy	UpdateStack
ValidateTemplate	

Amazon CloudFront

This guide focuses on the AWS SDK for PHP client for Amazon CloudFront. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudFront\CloudFrontClient::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\CloudFront\CloudFrontClient;

$client = CloudFrontClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon CloudFront is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;
```

```
// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudFront');
```

Signing CloudFront URLs for Private Distributions

Signed URLs allow you to provide users access to your private content. A signed URL includes additional information (e.g., expiration time) that gives you more control over access to your content. This additional information appears in a policy statement, which is based on either a canned policy or a custom policy. For information about how to set up private distributions and why you need to sign URLs, please read the Serving Private Content through CloudFront section of the CloudFront Developer Guide.

You can sign a URL using the CloudFront client in the SDK. First you must make sure to provide your CloudFront Private Key and Key Pair ID to the CloudFront client.

```
<?php

$cloudFront = CloudFrontClient::factory(array(
    'private_key' => '/path/to/your/cloudfront-private-key.pem',
    'key_pair_id' => '<cloudfront key pair id>',
));
```

You can alternatively specify the Private Key and Key Pair ID in your AWS config file and use the service builder to instantiate the CloudFront client. The following is an example config file that specifies the CloudFront key information.

```
<?php return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => '<aws access key>',
                'secret' => '<aws secret key>',
                'region' => 'us-west-2'
            )
        ),
        'cloudfront' => array(
            'extends' => 'cloudfront',
            'params'  => array(
                'private_key' => '/path/to/your/cloudfront-private-key.pem',
                'key_pair_id' => '<cloudfront key pair id>'
            )
        )
    )
);
```

You can sign a CloudFront URL for a video resource using either a canned or custom policy.

```
// Setup parameter values for the resource
$streamHostUrl = 'rtmp://example-distribution.cloudfront.net';
$resourceKey = 'videos/example.mp4';
$expires = time() + 300;

// Create a signed URL for the resource using the canned policy
$signedUrlCannedPolicy = $cloudFront->getSignedUrl(array(
    'url'     => $streamHostUrl . '/' . $resourceKey,
    'expires' => $expires,
));
```

For versions of the SDK later than 2.3.1, instead of providing your private key information when you instantiate the client, you can provide it at the time when you sign the URL.

```
$signedUrlCannedPolicy = $cloudFront->getSignedUrl(array(
    'url'         => $streamHostUrl . '/' . $resourceKey,
    'expires'     => $expires,
    'private_key' => '/path/to/your/cloudfront-private-key.pem',
    'key_pair_id' => '<cloudfront key pair id>'
));
```

To use a custom policy, provide the `policy` key instead of `expires`.

```
$customPolicy = <<<POLICY
{
    "Statement": [
        {
            "Resource": "{$resourceKey}",
            "Condition": {
                "IpAddress": {"AWS:SourceIp": "{$_SERVER['REMOTE_ADDR']}/32"},
                "DateLessThan": {"AWS:EpochTime": {$expires}}
            }
        }
    ]
}
POLICY;

// Create a signed URL for the resource using a custom policy
$signedUrlCustomPolicy = $cloudFront->getSignedUrl(array(
    'url'    => $streamHostUrl . '/' . $resourceKey,
    'policy' => $customPolicy,
));
```

The form of the signed URL is actually different depending on if the URL you are signing is using the "http" or "rtmp" scheme. In the case of "http", the full, absolute URL is returned. For "rtmp", only the relative URL is returned for your convenience, because some players require the host and path to be provided as separate parameters.

The following is an example of how you could use the signed URL to construct a web page displaying a video using JWPlayer. The same type of technique would apply to other players like FlowPlayer, but will require different client-side code.

```
<html>
<head>
    <title>Amazon CloudFront Streaming Example</title>
    <script type="text/javascript" src="https://example.com/jwplayer.js"></script>
</head>
<body>
    <div id="video">The canned policy video will be here.</div>
    <script type="text/javascript">
        jwplayer('video').setup({
            file: "<?= $streamHostUrl ?>/cfx/st/<?= $signedUrlCannedPolicy ?>",
            width: "720",
            height: "480"
        });
    </script>
</body>
</html>
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in

ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon CloudFront Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CreateCloudFrontOriginAccessIdentity	CreateDistribution
CreateInvalidation	CreateStreamingDistribution
DeleteCloudFrontOriginAccessIdentity	DeleteDistribution
DeleteStreamingDistribution	GetCloudFrontOriginAccessIdentity
GetCloudFrontOriginAccessIdentityConfig	GetDistribution
GetDistributionConfig	GetInvalidation
GetStreamingDistribution	GetStreamingDistributionConfig
ListCloudFrontOriginAccessIdentities	ListDistributions
ListInvalidations	ListStreamingDistributions
UpdateCloudFrontOriginAccessIdentity	UpdateDistribution
UpdateStreamingDistribution	

Amazon CloudFront (2012-05-05)

This guide focuses on the AWS SDK for PHP client for Amazon CloudFront. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Note: This guide is for the **2012-05-05** API version of Amazon CloudFront. You may also be interested in the *guide for the latest API version of Amazon CloudFront*.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudFront\CloudFrontClient::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\CloudFront\CloudFrontClient;

$client = CloudFrontClient::factory(array(
    'key'     => '<aws access key>',
    'secret'  => '<aws secret key>',
    'version' => '2012-05-05'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon CloudFront is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudFront');
```

Signing CloudFront URLs for Private Distributions

Signed URLs allow you to provide users access to your private content. A signed URL includes additional information (e.g., expiration time) that gives you more control over access to your content. This additional information appears in a policy statement, which is based on either a canned policy or a custom policy. For information about how to set up private distributions and why you need to sign URLs, please read the Serving Private Content through CloudFront section of the CloudFront Developer Guide.

You can sign a URL using the CloudFront client in the SDK. First you must make sure to provide your CloudFront Private Key and Key Pair ID to the CloudFront client.

```
<?php

$cloudFront = CloudFrontClient::factory(array(
    'private_key' => '/path/to/your/cloudfront-private-key.pem',
    'key_pair_id' => '<cloudfront key pair id>',
));
```

You can alternatively specify the Private Key and Key Pair ID in your AWS config file and use the service builder to instantiate the CloudFront client. The following is an example config file that specifies the CloudFront key information.

```
<?php return array(
    'includes' => array('_aws'),
    'services' => array(
        'default_settings' => array(
            'params' => array(
                'key'    => '<aws access key>',
                'secret' => '<aws secret key>',
                'region' => 'us-west-2'
            )
        ),
        'cloudfront' => array(
            'extends' => 'cloudfront',
            'params'  => array(
                'private_key' => '/path/to/your/cloudfront-private-key.pem',
                'key_pair_id' => '<cloudfront key pair id>'
            )
        )
    )
);
```

You can sign a CloudFront URL for a video resource using either a canned or custom policy.

```
// Setup parameter values for the resource
$streamHostUrl = 'rtmp://example-distribution.cloudfront.net';
$resourceKey = 'videos/example.mp4';
$expires = time() + 300;

// Create a signed URL for the resource using the canned policy
$signedUrlCannedPolicy = $cloudFront->getSignedUrl(array(
```

```
    'url'     => $streamHostUrl . '/' . $resourceKey,
    'expires' => $expires,
));
```

For versions of the SDK later than 2.3.1, instead of providing your private key information when you instantiate the client, you can provide it at the time when you sign the URL.

```
$signedUrlCannedPolicy = $cloudFront->getSignedUrl(array(
    'url'         => $streamHostUrl . '/' . $resourceKey,
    'expires'     => $expires,
    'private_key' => '/path/to/your/cloudfront-private-key.pem',
    'key_pair_id' => '<cloudfront key pair id>'
));
```

To use a custom policy, provide the `policy` key instead of `expires`.

```
$customPolicy = <<<POLICY
{
    "Statement": [
        {
            "Resource": "{$resourceKey}",
            "Condition": {
                "IpAddress": {"AWS:SourceIp": "{$_SERVER['REMOTE_ADDR']}/32"},
                "DateLessThan": {"AWS:EpochTime": {$expires}}
            }
        }
    ]
}
POLICY;

// Create a signed URL for the resource using a custom policy
$signedUrlCustomPolicy = $cloudFront->getSignedUrl(array(
    'url'    => $streamHostUrl . '/' . $resourceKey,
    'policy' => $customPolicy,
));
```

The form of the signed URL is actually different depending on if the URL you are signing is using the "http" or "rtmp" scheme. In the case of "http", the full, absolute URL is returned. For "rtmp", only the relative URL is returned for your convenience, because some players require the host and path to be provided as separate parameters.

The following is an example of how you could use the signed URL to construct a web page displaying a video using JWPlayer. The same type of technique would apply to other players like FlowPlayer, but will require different client-side code.

```
<html>
<head>
    <title>Amazon CloudFront Streaming Example</title>
    <script type="text/javascript" src="https://example.com/jwplayer.js"></script>
</head>
<body>
    <div id="video">The canned policy video will be here.</div>
    <script type="text/javascript">
        jwplayer('video').setup({
            file: "<?= $streamHostUrl ?>/cfx/st/<?= $signedUrlCannedPolicy ?>",
            width: "720",
            height: "480"
        });
    </script>
</body>
</html>
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon CloudFront Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CreateCloudFrontOriginAccessIdentity	CreateDistribution
CreateInvalidation	CreateStreamingDistribution
DeleteCloudFrontOriginAccessIdentity	DeleteDistribution
DeleteStreamingDistribution	GetCloudFrontOriginAccessIdentity
GetCloudFrontOriginAccessIdentityConfig	GetDistribution
GetDistributionConfig	GetInvalidation
GetStreamingDistribution	GetStreamingDistributionConfig
ListCloudFrontOriginAccessIdentities	ListDistributions
ListInvalidations	ListStreamingDistributions
UpdateCloudFrontOriginAccessIdentity	UpdateDistribution
UpdateStreamingDistribution	

Amazon CloudSearch

This guide focuses on the AWS SDK for PHP client for Amazon CloudSearch. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudSearch\CloudSearchClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-west-2`, `us-east-1`, `us-west-1`, `ap-southeast-1`, `eu-west-1`

```
use Aws\CloudSearch\CloudSearchClient;

$client = CloudSearchClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon CloudSearch is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudSearch');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon CloudSearch Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CreateDomain	DefineIndexField
DefineRankExpression	DeleteDomain
DeleteIndexField	DeleteRankExpression
DescribeDefaultSearchField	DescribeDomains
DescribeIndexFields	DescribeRankExpressions
DescribeServiceAccessPolicies	DescribeStemmingOptions
DescribeStopwordOptions	DescribeSynonymOptions
IndexDocuments	UpdateDefaultSearchField
UpdateServiceAccessPolicies	UpdateStemmingOptions
UpdateStopwordOptions	UpdateSynonymOptions

AWS CloudTrail

This guide focuses on the AWS SDK for PHP client for AWS CloudTrail. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudTrail\CloudTrailClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-west-2`, `us-east-1`

```
use Aws\CloudTrail\CloudTrailClient;

$client = CloudTrailClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS CloudTrail is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudTrail');
```

Blog articles

- Using AWS CloudTrail in PHP - Part 1
- Using AWS CloudTrail in PHP - Part 2

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS CloudTrail Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CreateTrail	DeleteTrail
DescribeTrails	GetTrailStatus
StartLogging	StopLogging
UpdateTrail	

Amazon CloudWatch

This guide focuses on the AWS SDK for PHP client for Amazon CloudWatch. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\CloudWatch\CloudWatchClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\CloudWatch\CloudWatchClient;

$client = CloudWatchClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon CloudWatch is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('CloudWatch');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon CloudWatch Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

DeleteAlarms	DescribeAlarmHistory

DescribeAlarms	DescribeAlarmsForMetric
DisableAlarmActions	EnableAlarmActions
GetMetricStatistics	ListMetrics
PutMetricAlarm	PutMetricData
SetAlarmState	

AWS Data Pipeline

This guide focuses on the AWS SDK for PHP client for AWS Data Pipeline. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\DataPipeline\DataPipelineClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`

```
use Aws\DataPipeline\DataPipelineClient;

$client = DataPipelineClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Data Pipeline is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('DataPipeline');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a

pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Data Pipeline Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

ActivatePipeline	CreatePipeline
DeletePipeline	DescribeObjects
DescribePipelines	EvaluateExpression
GetPipelineDefinition	ListPipelines
PollForTask	PutPipelineDefinition
QueryObjects	ReportTaskProgress
ReportTaskRunnerHeartbeat	SetStatus
SetTaskStatus	ValidatePipelineDefinition

AWS Direct Connect

This guide focuses on the AWS SDK for PHP client for AWS Direct Connect. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\DirectConnect\DirectConnectClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `eu-west-1`

```
use Aws\DirectConnect\DirectConnectClient;

$client = DirectConnectClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Direct Connect is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
```

```
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('DirectConnect');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Direct Connect Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AllocateConnectionOnInterconnect	AllocatePrivateVirtualInterface
AllocatePublicVirtualInterface	ConfirmConnection
ConfirmPrivateVirtualInterface	ConfirmPublicVirtualInterface
CreateConnection	CreateInterconnect
CreatePrivateVirtualInterface	CreatePublicVirtualInterface
DeleteConnection	DeleteInterconnect
DeleteVirtualInterface	DescribeConnections
DescribeConnectionsOnInterconnect	DescribeInterconnects
DescribeLocations	DescribeVirtualGateways
DescribeVirtualInterfaces	

Amazon DynamoDB

This guide focuses on the AWS SDK for PHP client for Amazon DynamoDB. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\DynamoDb\DynamoDbClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\DynamoDb\DynamoDbClient;

$client = DynamoDbClient::factory(array(
```

```
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon DynamoDB is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('DynamoDb');
```

Creating tables

You must first create a table that can be used to store items. Even though Amazon DynamoDB tables do not use a fixed schema, you do need to create a schema for the table's keys. This is explained in greater detail in Amazon DynamoDB's Data Model documentation. You will also need to specify the amount of provisioned throughput that should be made available to the table.

```
// Create an "errors" table
$client->createTable(array(
    'TableName' => 'errors',
    'AttributeDefinitions' => array(
        array(
            'AttributeName' => 'id',
            'AttributeType' => 'N'
        ),
        array(
            'AttributeName' => 'time',
            'AttributeType' => 'N'
        )
    ),
    'KeySchema' => array(
        array(
            'AttributeName' => 'id',
            'KeyType'       => 'HASH'
        ),
        array(
            'AttributeName' => 'time',
            'KeyType'       => 'RANGE'
        )
    ),
    'ProvisionedThroughput' => array(
        'ReadCapacityUnits'  => 10,
        'WriteCapacityUnits' => 20
    )
));
```

The table will now have a status of `CREATING` while the table is being provisioned. You can use a waiter to poll the table until it becomes `ACTIVE`.

```
// Wait until the table is created and active
$client->waitUntilTableExists(array(
    'TableName' => 'errors'
));
```

A full list of the parameters available to the `createTable()` operation can be found in the API documentation. For more information about using Local Secondary Indexes, please see the *Local secondary indexes* section of this guide.

Updating a table

You can also update the table after it's been created using the updateTable() method. This allows you to do things like increase or decrease your provisioned throughput capacity.

```
// Update the provisioned throughput capacity of the table
$client->updateTable(array(
    'TableName' => 'errors',
    'ProvisionedThroughput' => array(
        'ReadCapacityUnits'  => 15,
        'WriteCapacityUnits' => 25
    )
));

// Wait until the table is active again after updating
$client->waitUntilTableExists(array(
    'TableName' => 'errors'
));
```

Describing a table

Now that the table is created, you can use the describeTable() method to get information about the table.

```
$result = $client->describeTable(array(
    'TableName' => 'errors'
));

// The result of an operation can be used like an array
echo $result['Table']['ItemCount'] . "\n";
//> 0

// Use the getPath() method to retrieve deeply nested array key values
echo $result->getPath('Table/ProvisionedThroughput/ReadCapacityUnits') . "\n";
//> 15
```

The return value of the `describeTable()` method is a `Guzzle\Service\Resource\Model` object that can be used like an array. For example, you could retrieve the number of items in a table or the amount of provisioned read throughput.

Listing tables

You can retrieve a list of all of the tables associated with a specific endpoint using the listTables() method. Each Amazon DynamoDB endpoint is entirely independent. For example, if you have two tables called "MyTable," one in US-EAST-1 and one in US-WEST-2, they are completely independent and do not share any data. The ListTables operation returns all of the table names associated with the account making the request, for the endpoint that receives the request.

```
$result = $client->listTables();

// TableNames contains an array of table names
foreach ($result['TableNames'] as $tableName) {
```

```
    echo $tableName . "\n";
}
```

Iterating over all tables

The result of a `listTables()` operation might be truncated. Because of this, it is usually better to use an iterator to retrieve a complete list of all of the tables owned by your account in a specific region. The iterator will automatically handle sending any necessary subsequent requests.

```
$iterator = $client->getIterator('ListTables');

foreach ($iterator as $tableName) {
    echo $tableName . "\n";
}
```

> ***Tip***
>
> You can convert an iterator to an array using the `toArray()` method of the iterator.

Adding items

You can add an item to our *errors* table using the putItem() method of the client.

```
$time = time();

$result = $client->putItem(array(
    'TableName' => 'errors',
    'Item' => $client->formatAttributes(array(
        'id'      => 1201,
        'time'    => $time,
        'error'   => 'Executive overflow',
        'message' => 'no vacant areas'
    )),
    'ReturnConsumedCapacity' => 'TOTAL'
));

// The result will always contain ConsumedCapacityUnits
echo $result->getPath('ConsumedCapacity/CapacityUnits') . "\n";
```

As you can see, the `formatAttributes()` method of the client can be used to more easily format the attributes of the item. Alternatively, you can provide the item attributes without using the helper method:

```
$result = $client->putItem(array(
    'TableName' => 'errors',
    'Item' => array(
        'id'      => array('N' => '1201'),
        'time'    => array('N' => $time),
        'error'   => array('S' => 'Executive overflow'),
        'message' => array('S' => 'no vacant areas')
    )
));
```

You can also add items in batches of up to 25 items using the BatchWriteItem() method. Please see the example as shown in the *Local secondary indexes* section of this guide.

There is also a higher-level abstraction in the SDK over the `BatchWriteItem` operation called the `WriteRequestBatch` that handles queuing of write requests and retrying of unprocessed items. Please see the *Using the WriteRequestBatch* section of this guide for more information.

Retrieving items

You can check if the item was added correctly using the getItem() method of the client. Because Amazon DynamoDB works under an 'eventual consistency' model, we need to specify that we are performing a consistent read operation.

```
$result = $client->getItem(array(
    'ConsistentRead' => true,
    'TableName' => 'errors',
    'Key'       => array(
        'id'   => array('N' => '1201'),
        'time' => array('N' => $time)
    )
));

// Grab value from the result object like an array
echo $result['Item']['id']['N'] . "\n";
//> 1201
echo $result->getPath('Item/id/N') . "\n";
//> 1201
echo $result['Item']['error']['S'] . "\n";
//> Executive overflow
echo $result['Item']['message']['S'] . "\n";
//> no vacant areas
```

You can also retrieve items in batches of up to 100 using the BatchGetItem() method.

```
$tableName = 'errors';
$keys = array();

// Given that $keyValues contains a list of your hash and range keys:
//     array(array(<hash>, <range>), ...)
// Build the array for the "Keys" parameter
foreach ($keyValues as $values) {
    list($hashKeyValue, $rangeKeyValue) = $values;
    $keys[] = array(
        'id'   => array('N' => $hashKeyValue),
        'time' => array('N' => $rangeKeyValue)
    );
}

// Get multiple items by key in a BatchGetItem request
$result = $client->batchGetItem(array(
    'RequestItems' => array(
        $tableName => array(
            'Keys'           => $keys,
            'ConsistentRead' => true
        )
    )
));
$items = $result->getPath("Responses/{$tableName}");
```

Query and scan

Once data is in an Amazon DynamoDB table, you have two APIs for searching the data: Query and Scan.

Query

A query operation searches only primary key attribute values and supports a subset of comparison operators on key attribute values to refine the search process. A query returns all of the item data for the matching primary keys (all of each item's attributes) up to 1MB of data per query operation.

Let's say we want a list of all "1201" errors that occurred in the last 15 minutes. We could issue a single query that will search by the primary key of the table and retrieve up to 1MB of the items. However, a better approach is to use the query iterator to retrieve the entire list of all items matching the query.

```
$iterator = $client->getIterator('Query', array(
    'TableName'     => 'errors',
    'KeyConditions' => array(
        'id' => array(
            'AttributeValueList' => array(
                array('N' => '1201')
            ),
            'ComparisonOperator' => 'EQ'
        ),
        'time' => array(
            'AttributeValueList' => array(
                array('N' => strtotime("-15 minutes"))
            ),
            'ComparisonOperator' => 'GT'
        )
    )
));

// Each item will contain the attributes we added
foreach ($iterator as $item) {
    // Grab the time number value
    echo $item['time']['N'] . "\n";
    // Grab the error string value
    echo $item['error']['S'] . "\n";
}
```

Scan

A scan operation scans the entire table. You can specify filters to apply to the results to refine the values returned to you, after the complete scan. Amazon DynamoDB puts a 1MB limit on the scan (the limit applies before the results are filtered).

A scan can be useful for more complex searches. For example, we can retrieve all of the errors in the last 15 minutes that contain the word "overflow":

```
$iterator = $client->getIterator('Scan', array(
    'TableName' => 'errors',
    'ScanFilter' => array(
        'error' => array(
            'AttributeValueList' => array(
                array('S' => 'overflow')
            ),
            'ComparisonOperator' => 'CONTAINS'
        ),
        'time' => array(
            'AttributeValueList' => array(
                array('N' => strtotime('-15 minutes'))
            ),
            'ComparisonOperator' => 'GT'
        )
    )
));

// Each item will contain the attributes we added
foreach ($iterator as $item) {
    // Grab the time number value
    echo $item['time']['N'] . "\n";
    // Grab the error string value
```

```
    echo $item['error']['S'] . "\n";
}
```

Deleting items

To delete an item you must use the DeleteItem() method. The following example scans through a table and deletes every item one by one.

```
$scan = $client->getIterator('Scan', array('TableName' => 'errors'));
foreach ($scan as $item) {
    $client->deleteItem(array(
        'TableName' => 'errors',
        'Key' => array(
            'id'   => array('N' => $item['id']['N']),
            'time' => array('N' => $item['time']['N'])
        )
    ));
}
```

You can also delete items in batches of up to 25 items using the BatchWriteItem() method.

Deleting a table

Warning

Deleting a table will also permanently delete all of its contents.

Now that you've taken a quick tour of the PHP client for Amazon DynamoDB, you will want to clean up by deleting the resources you created.

```
$client->deleteTable(array(
    'TableName' => 'errors'
));

$client->waitUntilTableNotExists(array(
    'TableName' => 'errors'
));
```

Local secondary indexes

Local secondary indexes (LSI) pair your table's leading hash key with an alternate range key, in order to enable specific queries to run more quickly than they would using a standard composite primary key. The following code samples will show how to create an *Orders* table with a hash key of *CustomerId* and a range key of *OrderId*, but also include a local secondary index on the *OrderDate* attribute so that searching the table based by *OrderDate* can be done with a `Query` operation instead of a `Scan` operation.

First you must create the table with the local secondary index. Note that the attributes referenced in the key schema for the table *and* the index must all be declared in the `AttributeDefinitions` parameter. When you create a local secondary index, you can specify which attributes get "projected" into the index using the `Projection` parameter.

```
// Create an "Orders" table
$client->createTable(array(
    'TableName' => 'Orders',
    'AttributeDefinitions' => array(
        array('AttributeName' => 'CustomerId', 'AttributeType' => 'N'),
        array('AttributeName' => 'OrderId',    'AttributeType' => 'N'),
        array('AttributeName' => 'OrderDate',  'AttributeType' => 'N'),
    ),
```

```
    'KeySchema' => array(
        array('AttributeName' => 'CustomerId', 'KeyType' => 'HASH'),
        array('AttributeName' => 'OrderId',    'KeyType' => 'RANGE'),
    ),
    'LocalSecondaryIndexes' => array(
        array(
            'IndexName' => 'OrderDateIndex',
            'KeySchema' => array(
                array('AttributeName' => 'CustomerId', 'KeyType' => 'HASH'),
                array('AttributeName' => 'OrderDate',  'KeyType' => 'RANGE'),
            ),
            'Projection' => array(
                'ProjectionType' => 'KEYS_ONLY',
            ),
        ),
    ),
    'ProvisionedThroughput' => array(
        'ReadCapacityUnits'  => 10,
        'WriteCapacityUnits' => 20
    )
));

$client->waitUntilTableExists(array('TableName' => 'Orders'));
```

Next you must add some items to the table that you will be querying. There's nothing in the `BatchWriteItem` operation that is specific to the LSI features, but since there is not an example of this operation elsewhere in the guide, this seems like a good place to show how to use this operation.

```
$result = $client->batchWriteItem(array(
    'RequestItems' => array(
        'Orders' => array(
            array(
                'PutRequest' => array(
                    'Item' => array(
                        'CustomerId' => array('N' => 1041),
                        'OrderId'    => array('N' => 6),
                        'OrderDate'  => array('N' => strtotime('-5 days')),
                        'ItemId'     => array('N' => 25336)
                    )
                )
            ),
            array(
                'PutRequest' => array(
                    'Item' => array(
                        'CustomerId' => array('N' => 941),
                        'OrderId'    => array('N' => 8),
                        'OrderDate'  => array('N' => strtotime('-3 days')),
                        'ItemId'     => array('N' => 15596)
                    )
                )
            ),
            array(
                'PutRequest' => array(
                    'Item' => array(
                        'CustomerId' => array('N' => 941),
                        'OrderId'    => array('N' => 2),
                        'OrderDate'  => array('N' => strtotime('-12 days')),
                        'ItemId'     => array('N' => 38449)
                    )
                )
            ),
```

```
            array(
                'PutRequest' => array(
                    'Item' => array(
                        'CustomerId' => array('N' => 941),
                        'OrderId'    => array('N' => 3),
                        'OrderDate'  => array('N' => strtotime('-1 days')),
                        'ItemId'     => array('N' => 25336)
                    )
                )
            )
        )
    )
));
```

When you query the table with an LSI, you must specify the name of the index using the `IndexName` parameter. The attributes that are returned will depend on the value of the `Select` parameter and on what the table is projecting to the index. In this case `'Select' => 'COUNT'` has been specified, so only the count of the items will be returned.

```
// Find the number of orders made by customer 941 in the last 10 days
$result = $client->query(array(
    'TableName'     => 'Orders',
    'IndexName'     => 'OrderDateIndex',
    'Select'        => 'COUNT',
    'KeyConditions' => array(
        'CustomerId' => array(
            'AttributeValueList' => array(
                array('N' => '941')
            ),
            'ComparisonOperator' => 'EQ'
        ),
        'OrderDate' => array(
            'AttributeValueList' => array(
                array('N' => strtotime("-10 days"))
            ),
            'ComparisonOperator' => 'GE'
        )
    )
));

$numOrders = $result['Count'];
```

Using the WriteRequestBatch

You can use the `WriteRequestBatch` if you need to write or delete many items as quickly as possible. The WriteRequestBatch provides a high level of performance because it converts what would normally be a separate HTTP request for each operation into HTTP requests containing up to 25 comparable requests per transaction.

If you have a large array of items you wish to add to your table, you could iterate over the them, add each item to the batch object. After all the items are added call `flush()`. The batch object will automatically flush the batch and write items to Amazon DynamoDB after hitting a customizable threshold. A final call to the batch object's `flush()` method is necessary to transfer any remaining items in the queue.

```
$tableName = 'batch-write-test'; // This table has a HashKey named "id"
$itemIds = array();

// Put 55 items into the table
$putBatch = WriteRequestBatch::factory($client);
for ($i = 0; $i < 55; $i++) {
    $itemIds[] = $itemId = uniqid();
    $item = Item::fromArray(array(
```

```
        'id'        => $itemId,
        'timestamp' => time(),
    ));
    $putBatch->add(new PutRequest($item, $tableName));
}
$putBatch->flush();
```

You can also use the `WriteRequestBatch` object to delete items in batches.

```
// Remove items from the table
$deleteBatch = WriteRequestBatch::factory($client);
foreach ($itemIds as $itemId) {
    $key = array('id' => array('S' => $itemId));
    $deleteBatch->add(new DeleteRequest($key, $tableName));
}
$deleteBatch->flush();
```

The `WriteRequestBatch`, `PutRequest`, and `DeleteRequest` classes are all a part of the `Aws\DynamoDb\Model\BatchRequest` namespace.

API Reference

Please see the Amazon DynamoDB Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

BatchGetItem	BatchWriteItem
CreateTable	DeleteItem
DeleteTable	DescribeTable
GetItem	ListTables
PutItem	Query
Scan	UpdateItem
UpdateTable	

Amazon DynamoDB (2011-12-05)

This guide focuses on the AWS SDK for PHP client for Amazon DynamoDB. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Note: This guide is for the **2011-12-05** API version of Amazon DynamoDB. You may also be interested in the *guide for the latest API version of Amazon DynamoDB*.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\DynamoDb\DynamoDbClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\DynamoDb\DynamoDbClient;

$client = DynamoDbClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
```

```
    'region'  => '<region name>',
    'version' => '2011-12-05'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon DynamoDB is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('DynamoDb');
```

Creating tables

You must first create a table that can be used to store items. Even though Amazon DynamoDB tables do not use a fixed schema, you do need to create a schema for the table's keys. This is explained in greater detail in Amazon DynamoDB's Data Model documentation. You will also need to specify the amount of provisioned throughput that should be made available to the table.

```
// Create an "errors" table
$client->createTable(array(
    'TableName' => 'errors',
    'KeySchema' => array(
        'HashKeyElement' => array(
            'AttributeName' => 'id',
            'AttributeType' => 'N'
        ),
        'RangeKeyElement' => array(
            'AttributeName' => 'time',
            'AttributeType' => 'N'
        )
    ),
    'ProvisionedThroughput' => array(
        'ReadCapacityUnits'  => 10,
        'WriteCapacityUnits' => 20
    )
));
```

The table will now have a status of `CREATING` while the table is being provisioned. You can use a waiter to poll the table until it becomes `ACTIVE`.

```
// Wait until the table is created and active
$client->waitUntilTableExists(array(
    'TableName' => 'errors'
));
```

A full list of the parameters available to the `createTable()` operation can be found in the API documentation.

Updating a table

You can also update the table after it's been created using the updateTable() method. This allows you to do things like increase or decrease your provisioned throughput capacity.

```
// Update the provisioned throughput capacity of the table
$client->updateTable(array(
    'TableName' => 'errors',
    'ProvisionedThroughput' => array(
        'ReadCapacityUnits'  => 15,
        'WriteCapacityUnits' => 25
    )
));

// Wait until the table is active again after updating
$client->waitUntilTableExists(array(
    'TableName' => 'errors'
));
```

Describing a table

Now that the table is created, you can use the describeTable() method to get information about the table.

```
$result = $client->describeTable(array(
    'TableName' => 'errors'
));

// The result of an operation can be used like an array
echo $result['Table']['ItemCount'] . "\n";
//> 0

// Use the getPath() method to retrieve deeply nested array key values
echo $result->getPath('Table/ProvisionedThroughput/ReadCapacityUnits') . "\n";
//> 15
```

The return value of the `describeTable()` method is a `Guzzle\Service\Resource\Model` object that can be used like an array. For example, you could retrieve the number of items in a table or the amount of provisioned read throughput.

Listing tables

You can retrieve a list of all of the tables associated with a specific endpoint using the listTables() method. Each Amazon DynamoDB endpoint is entirely independent. For example, if you have two tables called "MyTable," one in US-EAST-1 and one in US-WEST-2, they are completely independent and do not share any data. The ListTables operation returns all of the table names associated with the account making the request, for the endpoint that receives the request.

```
$result = $client->listTables();

// TableNames contains an array of table names
foreach ($result['TableNames'] as $tableName) {
    echo $tableName . "\n";
}
```

Iterating over all tables

The result of a `listTables()` operation might be truncated. Because of this, it is usually better to use an iterator to retrieve a complete list of all of the tables owned by your account in a specific region. The iterator will automatically handle sending any necessary subsequent requests.

```
$iterator = $client->getIterator('ListTables');

foreach ($iterator as $tableName) {
    echo $tableName . "\n";
}
```

> ***Tip***
>
> You can convert an iterator to an array using the `toArray()` method of the iterator.

Adding items

You can add an item to our *errors* table using the putItem() method of the client.

```
$time = time();

$result = $client->putItem(array(
    'TableName' => 'errors',
    'Item' => $client->formatAttributes(array(
        'id'      => 1201,
        'time'    => $time,
        'error'   => 'Executive overflow',
        'message' => 'no vacant areas'
    ))
));

// The result will always contain ConsumedCapacityUnits
echo $result['ConsumedCapacityUnits'] . "\n";
```

As you can see, the `formatAttributes()` method of the client can be used to more easily format the attributes of the item. Alternatively, you can provide the item attributes without using the helper method:

```
$result = $client->putItem(array(
    'TableName' => 'errors',
    'Item' => array(
        'id'      => array('N' => '1201'),
        'time'    => array('N' => $time),
        'error'   => array('S' => 'Executive overflow'),
        'message' => array('S' => 'no vacant areas')
    )
));
```

Retrieving items

You can check if the item was added correctly using the getItem() method of the client. Because Amazon DynamoDB works under an 'eventual consistency' model, we need to specify that we are performing a consistent read operation.

```
$result = $client->getItem(array(
    'ConsistentRead' => true,
    'TableName' => 'errors',
    'Key' => array(
        'HashKeyElement'  => array('N' => '1201'),
        'RangeKeyElement' => array('N' => $time)
    )
));

// Grab value from the result object like an array
echo $result['Item']['id']['N'] . "\n";
//> 1201
echo $result->getPath('Item/id/N') . "\n";
//> 1201
echo $result['Item']['error']['S'] . "\n";
//> Executive overflow
```

```
echo $result['Item']['message']['S'] . "\n";
//> no vacant areas
```

You can also retrieve items in batches of up to 100 using the BatchGetItem() method.

```
$tableName = 'errors';
$keys = array();

// Given that $keyValues contains a list of your hash and range keys:
//     array(array(<hash>, <range>), ...)
// Build the array for the "Keys" parameter
foreach ($keyValues as $values) {
    list($hashKeyValue, $rangeKeyValue) = $values;
    $keys[] = array(
        'HashKeyElement'  => array('N' => $hashKeyValue),
        'RangeKeyElement' => array('N' => $rangeKeyValue)
    );
}

// Get multiple items by key in a BatchGetItem request
$result = $client->batchGetItem(array(
    'RequestItems' => array(
        $tableName => array(
            'Keys'           => $keys,
            'ConsistentRead' => true
        )
    )
));
$items = $result->getPath("Responses/{$tableName}");
```

Query and scan

Once data is in an Amazon DynamoDB table, you have two APIs for searching the data: Query and Scan.

Query

A query operation searches only primary key attribute values and supports a subset of comparison operators on key attribute values to refine the search process. A query returns all of the item data for the matching primary keys (all of each item's attributes) up to 1MB of data per query operation.

Let's say we want a list of all "1201" errors that occurred in the last 15 minutes. We could issue a single query that will search by the primary key of the table and retrieve up to 1MB of the items. However, a better approach is to use the query iterator to retrieve the entire list of all items matching the query.

```
$iterator = $client->getIterator('Query', array(
    'TableName'         => 'errors',
    'HashKeyValue'      => array('N' => '1201'),
    'RangeKeyCondition' => array(
        'AttributeValueList' => array(
            array('N' => strtotime("-15 minutes"))
        ),
        'ComparisonOperator' => 'GT'
    )
));

// Each item will contain the attributes we added
foreach ($iterator as $item) {
    // Grab the time number value
    echo $item['time']['N'] . "\n";
    // Grab the error string value
```

```
    echo $item['error']['S'] . "\n";
}
```

Scan

A scan operation scans the entire table. You can specify filters to apply to the results to refine the values returned to you, after the complete scan. Amazon DynamoDB puts a 1MB limit on the scan (the limit applies before the results are filtered).

A scan can be useful for more complex searches. For example, we can retrieve all of the errors in the last 15 minutes that contain the word "overflow":

```
$iterator = $client->getIterator('Scan', array(
    'TableName' => 'errors',
    'ScanFilter' => array(
        'error' => array(
            'AttributeValueList' => array(array('S' => 'overflow')),
            'ComparisonOperator' => 'CONTAINS'
        ),
        'time' => array(
            'AttributeValueList' => array(
                array('N' => strtotime('-15 minutes'))
            ),
            'ComparisonOperator' => 'GT'
        )
    )
));

// Each item will contain the attributes we added
foreach ($iterator as $item) {
    // Grab the time number value
    echo $item['time']['N'] . "\n";
    // Grab the error string value
    echo $item['error']['S'] . "\n";
}
```

Deleting a table

> **Warning**
>
> Deleting a table will also permanently delete all of its contents.

Now that you've taken a quick tour of the PHP client for Amazon DynamoDB, you will want to clean up by deleting the resources you created.

```
$client->deleteTable(array(
    'TableName' => 'errors'
));

$client->waitUntilTableNotExists(array(
    'TableName' => 'errors'
));
```

Using the WriteRequestBatch

You can use the `WriteRequestBatch` if you need to write or delete many items as quickly as possible. The WriteRequestBatch provides a high level of performance because it converts what would normally be a separate HTTP request for each operation into HTTP requests containing up to 25 comparable requests per transaction.

If you have a large array of items you wish to add to your table, you could iterate over the them, add each item to the batch object. After all the items are added call `flush()`. The batch object will automatically flush the batch and write items to Amazon DynamoDB after hitting a customizable threshold. A final call to the batch object's `flush()` method is necessary to transfer any remaining items in the queue.

```
$tableName = 'batch-write-test'; // This table has a HashKey named "id"
$itemIds = array();

// Put 55 items into the table
$putBatch = WriteRequestBatch::factory($client);
for ($i = 0; $i < 55; $i++) {
    $itemIds[] = $itemId = uniqid();
    $item = Item::fromArray(array(
        'id'        => $itemId,
        'timestamp' => time(),
    ));
    $putBatch->add(new PutRequest($item, $tableName));
}
$putBatch->flush();
```

You can also use the `WriteRequestBatch` object to delete items in batches.

```
// Remove items from the table
$deleteBatch = WriteRequestBatch::factory($client);
foreach ($itemIds as $itemId) {
    $key = array('HashKeyElement' => array('S' => $itemId));
    $deleteBatch->add(new DeleteRequest($key, $tableName));
}
$deleteBatch->flush();
```

The `WriteRequestBatch`, `PutRequest`, and `DeleteRequest` classes are all a part of the `Aws\DynamoDb\Model\BatchRequest` namespace.

API Reference

Please see the Amazon DynamoDB Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

BatchGetItem	BatchWriteItem
CreateTable	DeleteItem
DeleteTable	DescribeTable
GetItem	ListTables
PutItem	Query
Scan	UpdateItem
UpdateTable	

Amazon Elastic Compute Cloud

This guide focuses on the AWS SDK for PHP client for Amazon Elastic Compute Cloud. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Ec2\Ec2Client::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Ec2\Ec2Client;

$client = Ec2Client::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Elastic Compute Cloud is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Ec2');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Elastic Compute Cloud Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

ActivateLicense	AllocateAddress
AssignPrivateIpAddresses	AssociateAddress
AssociateDhcpOptions	AssociateRouteTable
AttachInternetGateway	AttachNetworkInterface
AttachVolume	AttachVpnGateway
AuthorizeSecurityGroupEgress	AuthorizeSecurityGroupIngress
BundleInstance	CancelBundleTask

CancelConversionTask	CancelExportTask
CancelReservedInstancesListing	CancelSpotInstanceRequests
ConfirmProductInstance	CopyImage
CopySnapshot	CreateCustomerGateway
CreateDhcpOptions	CreateImage
CreateInstanceExportTask	CreateInternetGateway
CreateKeyPair	CreateNetworkAcl
CreateNetworkAclEntry	CreateNetworkInterface
CreatePlacementGroup	CreateReservedInstancesListing
CreateRoute	CreateRouteTable
CreateSecurityGroup	CreateSnapshot
CreateSpotDatafeedSubscription	CreateSubnet
CreateTags	CreateVolume
CreateVpc	CreateVpnConnection
CreateVpnConnectionRoute	CreateVpnGateway
DeactivateLicense	DeleteCustomerGateway
DeleteDhcpOptions	DeleteInternetGateway
DeleteKeyPair	DeleteNetworkAcl
DeleteNetworkAclEntry	DeleteNetworkInterface
DeletePlacementGroup	DeleteRoute
DeleteRouteTable	DeleteSecurityGroup
DeleteSnapshot	DeleteSpotDatafeedSubscription
DeleteSubnet	DeleteTags
DeleteVolume	DeleteVpc
DeleteVpnConnection	DeleteVpnConnectionRoute
DeleteVpnGateway	DeregisterImage
DescribeAccountAttributes	DescribeAddresses
DescribeAvailabilityZones	DescribeBundleTasks
DescribeConversionTasks	DescribeCustomerGateways
DescribeDhcpOptions	DescribeExportTasks
DescribeImageAttribute	DescribeImages
DescribeInstanceAttribute	DescribeInstanceStatus
DescribeInstances	DescribeInternetGateways
DescribeKeyPairs	DescribeLicenses
DescribeNetworkAcls	DescribeNetworkInterfaceAttribute
DescribeNetworkInterfaces	DescribePlacementGroups
DescribeRegions	DescribeReservedInstances
DescribeReservedInstancesListings	DescribeReservedInstancesModifications
DescribeReservedInstancesOfferings	DescribeRouteTables
DescribeSecurityGroups	DescribeSnapshotAttribute

DescribeSnapshots	DescribeSpotDatafeedSubscription
DescribeSpotInstanceRequests	DescribeSpotPriceHistory
DescribeSubnets	DescribeTags
DescribeVolumeAttribute	DescribeVolumeStatus
DescribeVolumes	DescribeVpcAttribute
DescribeVpcs	DescribeVpnConnections
DescribeVpnGateways	DetachInternetGateway
DetachNetworkInterface	DetachVolume
DetachVpnGateway	DisableVgwRoutePropagation
DisassociateAddress	DisassociateRouteTable
EnableVgwRoutePropagation	EnableVolumeIO
GetConsoleOutput	GetPasswordData
ImportInstance	ImportKeyPair
ImportVolume	ModifyImageAttribute
ModifyInstanceAttribute	ModifyNetworkInterfaceAttribute
ModifyReservedInstances	ModifySnapshotAttribute
ModifyVolumeAttribute	ModifyVpcAttribute
MonitorInstances	PurchaseReservedInstancesOffering
RebootInstances	RegisterImage
ReleaseAddress	ReplaceNetworkAclAssociation
ReplaceNetworkAclEntry	ReplaceRoute
ReplaceRouteTableAssociation	ReportInstanceStatus
RequestSpotInstances	ResetImageAttribute
ResetInstanceAttribute	ResetNetworkInterfaceAttribute
ResetSnapshotAttribute	RevokeSecurityGroupEgress
RevokeSecurityGroupIngress	RunInstances
StartInstances	StopInstances
TerminateInstances	UnassignPrivateIpAddresses
UnmonitorInstances	

Amazon ElastiCache

This guide focuses on the AWS SDK for PHP client for Amazon ElastiCache. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\ElastiCache\ElastiCacheClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\ElastiCache\ElastiCacheClient;

$client = ElastiCacheClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon ElastiCache is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('ElastiCache');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon ElastiCache Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AuthorizeCacheSecurityGroupIngress	CreateCacheCluster
CreateCacheParameterGroup	CreateCacheSecurityGroup
CreateCacheSubnetGroup	CreateReplicationGroup
DeleteCacheCluster	DeleteCacheParameterGroup
DeleteCacheSecurityGroup	DeleteCacheSubnetGroup
DeleteReplicationGroup	DescribeCacheClusters
DescribeCacheEngineVersions	DescribeCacheParameterGroups
DescribeCacheParameters	DescribeCacheSecurityGroups
DescribeCacheSubnetGroups	DescribeEngineDefaultParameters

DescribeEvents	DescribeReplicationGroups
DescribeReservedCacheNodes	DescribeReservedCacheNodesOfferings
ModifyCacheCluster	ModifyCacheParameterGroup
ModifyCacheSubnetGroup	ModifyReplicationGroup
PurchaseReservedCacheNodesOffering	RebootCacheCluster
ResetCacheParameterGroup	RevokeCacheSecurityGroupIngress

AWS Elastic Beanstalk

This guide focuses on the AWS SDK for PHP client for AWS Elastic Beanstalk. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\ElasticBeanstalk\ElasticBeanstalkClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `eu-west-1`

```
use Aws\ElasticBeanstalk\ElasticBeanstalkClient;

$client = ElasticBeanstalkClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Elastic Beanstalk is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('ElasticBeanstalk');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in

ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Elastic Beanstalk Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CheckDNSAvailability	CreateApplication
CreateApplicationVersion	CreateConfigurationTemplate
CreateEnvironment	CreateStorageLocation
DeleteApplication	DeleteApplicationVersion
DeleteConfigurationTemplate	DeleteEnvironmentConfiguration
DescribeApplicationVersions	DescribeApplications
DescribeConfigurationOptions	DescribeConfigurationSettings
DescribeEnvironmentResources	DescribeEnvironments
DescribeEvents	ListAvailableSolutionStacks
RebuildEnvironment	RequestEnvironmentInfo
RestartAppServer	RetrieveEnvironmentInfo
SwapEnvironmentCNAMEs	TerminateEnvironment
UpdateApplication	UpdateApplicationVersion
UpdateConfigurationTemplate	UpdateEnvironment
ValidateConfigurationSettings	

Elastic Load Balancing

This guide focuses on the AWS SDK for PHP client for Elastic Load Balancing. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\ElasticLoadBalancing\ElasticLoadBalancingClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\ElasticLoadBalancing\ElasticLoadBalancingClient;

$client = ElasticLoadBalancingClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Elastic Load Balancing is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('ElasticLoadBalancing');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Elastic Load Balancing Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

ApplySecurityGroupsToLoadBalancer	AttachLoadBalancerToSubnets
ConfigureHealthCheck	CreateAppCookieStickinessPolicy
CreateLBCookieStickinessPolicy	CreateLoadBalancer
CreateLoadBalancerListeners	CreateLoadBalancerPolicy
DeleteLoadBalancer	DeleteLoadBalancerListeners
DeleteLoadBalancerPolicy	DeregisterInstancesFromLoadBalancer
DescribeInstanceHealth	DescribeLoadBalancerAttributes
DescribeLoadBalancerPolicies	DescribeLoadBalancerPolicyTypes
DescribeLoadBalancers	DetachLoadBalancerFromSubnets
DisableAvailabilityZonesForLoadBalancer	EnableAvailabilityZonesForLoadBalancer
ModifyLoadBalancerAttributes	RegisterInstancesWithLoadBalancer
SetLoadBalancerListenerSSLCertificate	SetLoadBalancerPoliciesForBackendServer
SetLoadBalancerPoliciesOfListener	

Amazon Elastic Transcoder

This guide focuses on the AWS SDK for PHP client for Amazon Elastic Transcoder. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\ElasticTranscoder\ElasticTranscoderClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `eu-west-1`

```
use Aws\ElasticTranscoder\ElasticTranscoderClient;

$client = ElasticTranscoderClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Elastic Transcoder is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('ElasticTranscoder');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Elastic Transcoder Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CancelJob	CreateJob
CreatePipeline	CreatePreset
DeletePipeline	DeletePreset
ListJobsByPipeline	ListJobsByStatus
ListPipelines	ListPresets
ReadJob	ReadPipeline
ReadPreset	TestRole
UpdatePipeline	UpdatePipelineNotifications
UpdatePipelineStatus	

Amazon Elastic MapReduce

This guide focuses on the AWS SDK for PHP client for Amazon Elastic MapReduce. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Emr\EmrClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Emr\EmrClient;

$client = EmrClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Elastic MapReduce is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Emr');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Elastic MapReduce Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddInstanceGroups	AddJobFlowSteps
AddTags	DescribeCluster
DescribeJobFlows	DescribeStep
ListBootstrapActions	ListClusters
ListInstanceGroups	ListInstances
ListSteps	ModifyInstanceGroups
RemoveTags	RunJobFlow
SetTerminationProtection	SetVisibleToAllUsers
TerminateJobFlows	

Amazon Glacier

This guide focuses on the AWS SDK for PHP client for Amazon Glacier. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Glacier\GlacierClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Glacier\GlacierClient;

$client = GlacierClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Glacier is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Glacier');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Glacier Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AbortMultipartUpload	CompleteMultipartUpload
CreateVault	DeleteArchive
DeleteVault	DeleteVaultNotifications
DescribeJob	DescribeVault
GetJobOutput	GetVaultNotifications
InitiateJob	InitiateMultipartUpload
ListJobs	ListMultipartUploads
ListParts	ListVaults
SetVaultNotifications	UploadArchive
UploadMultipartPart	

AWS Identity and Access Management

This guide focuses on the AWS SDK for PHP client for AWS Identity and Access Management. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Iam\IamClient::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\Iam\IamClient;

$client = IamClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Identity and Access Management is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Iam');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Identity and Access Management Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddRoleToInstanceProfile	AddUserToGroup
ChangePassword	CreateAccessKey
CreateAccountAlias	CreateGroup
CreateInstanceProfile	CreateLoginProfile
CreateRole	CreateSAMLProvider
CreateUser	CreateVirtualMFADevice
DeactivateMFADevice	DeleteAccessKey
DeleteAccountAlias	DeleteAccountPasswordPolicy
DeleteGroup	DeleteGroupPolicy
DeleteInstanceProfile	DeleteLoginProfile
DeleteRole	DeleteRolePolicy
DeleteSAMLProvider	DeleteServerCertificate

DeleteSigningCertificate	DeleteUser
DeleteUserPolicy	DeleteVirtualMFADevice
EnableMFADevice	GetAccountPasswordPolicy
GetAccountSummary	GetGroup
GetGroupPolicy	GetInstanceProfile
GetLoginProfile	GetRole
GetRolePolicy	GetSAMLProvider
GetServerCertificate	GetUser
GetUserPolicy	ListAccessKeys
ListAccountAliases	ListGroupPolicies
ListGroups	ListGroupsForUser
ListInstanceProfiles	ListInstanceProfilesForRole
ListMFADevices	ListRolePolicies
ListRoles	ListSAMLProviders
ListServerCertificates	ListSigningCertificates
ListUserPolicies	ListUsers
ListVirtualMFADevices	PutGroupPolicy
PutRolePolicy	PutUserPolicy
RemoveRoleFromInstanceProfile	RemoveUserFromGroup
ResyncMFADevice	UpdateAccessKey
UpdateAccountPasswordPolicy	UpdateAssumeRolePolicy
UpdateGroup	UpdateLoginProfile
UpdateSAMLProvider	UpdateServerCertificate
UpdateSigningCertificate	UpdateUser
UploadServerCertificate	UploadSigningCertificate

AWS Import/Export

This guide focuses on the AWS SDK for PHP client for AWS Import/Export. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\ImportExport\ImportExportClient::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\ImportExport\ImportExportClient;

$client = ImportExportClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Import/Export is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('ImportExport');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Import/Export Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CancelJob	CreateJob
GetStatus	ListJobs
UpdateJob	

Amazon Kinesis

This guide focuses on the AWS SDK for PHP client for Amazon Kinesis. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Kinesis\KinesisClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`

```
use Aws\Kinesis\KinesisClient;

$client = KinesisClient::factory(array(
    'key'    => '<aws access key>',
```

```
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Kinesis is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Kinesis');
```

To learn about the Amazon Kinesis API, see the Amazon Kinesis Developer Guide.

Creating a stream

The main resource in Amazon Kinesis is the stream. To create a stream you need to decide on a stream name and shard count.

```
$client->createStream(array(
    'StreamName' => 'php-test-stream',
    'ShardCount' => 1,
));
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Kinesis Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CreateStream	DeleteStream
DescribeStream	GetRecords
GetShardIterator	ListStreams
MergeShards	PutRecord
SplitShard	

AWS OpsWorks

This guide focuses on the AWS SDK for PHP client for AWS OpsWorks. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\OpsWorks\OpsWorksClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`

```
use Aws\OpsWorks\OpsWorksClient;

$client = OpsWorksClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS OpsWorks is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('OpsWorks');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS OpsWorks Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AssignVolume	AssociateElasticIp

AttachElasticLoadBalancer	CloneStack
CreateApp	CreateDeployment
CreateInstance	CreateLayer
CreateStack	CreateUserProfile
DeleteApp	DeleteInstance
DeleteLayer	DeleteStack
DeleteUserProfile	DeregisterElasticIp
DeregisterVolume	DescribeApps
DescribeCommands	DescribeDeployments
DescribeElasticIps	DescribeElasticLoadBalancers
DescribeInstances	DescribeLayers
DescribeLoadBasedAutoScaling	DescribeMyUserProfile
DescribePermissions	DescribeRaidArrays
DescribeServiceErrors	DescribeStackSummary
DescribeStacks	DescribeTimeBasedAutoScaling
DescribeUserProfiles	DescribeVolumes
DetachElasticLoadBalancer	DisassociateElasticIp
GetHostnameSuggestion	RebootInstance
RegisterElasticIp	RegisterVolume
SetLoadBasedAutoScaling	SetPermission
SetTimeBasedAutoScaling	StartInstance
StartStack	StopInstance
StopStack	UnassignVolume
UpdateApp	UpdateElasticIp
UpdateInstance	UpdateLayer
UpdateMyUserProfile	UpdateStack
UpdateUserProfile	UpdateVolume

Amazon Relational Database Service

This guide focuses on the AWS SDK for PHP client for Amazon Relational Database Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Rds\RdsClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Rds\RdsClient;

$client = RdsClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Relational Database Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Rds');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Relational Database Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddSourceIdentifierToSubscription	AddTagsToResource
AuthorizeDBSecurityGroupIngress	CopyDBSnapshot
CreateDBInstance	CreateDBInstanceReadReplica
CreateDBParameterGroup	CreateDBSecurityGroup
CreateDBSnapshot	CreateDBSubnetGroup
CreateEventSubscription	CreateOptionGroup
DeleteDBInstance	DeleteDBParameterGroup
DeleteDBSecurityGroup	DeleteDBSnapshot
DeleteDBSubnetGroup	DeleteEventSubscription
DeleteOptionGroup	DescribeDBEngineVersions
DescribeDBInstances	DescribeDBLogFiles

DescribeDBParameterGroups	DescribeDBParameters
DescribeDBSecurityGroups	DescribeDBSnapshots
DescribeDBSubnetGroups	DescribeEngineDefaultParameters
DescribeEventCategories	DescribeEventSubscriptions
DescribeEvents	DescribeOptionGroupOptions
DescribeOptionGroups	DescribeOrderableDBInstanceOptions
DescribeReservedDBInstances	DescribeReservedDBInstancesOfferings
DownloadDBLogFilePortion	ListTagsForResource
ModifyDBInstance	ModifyDBParameterGroup
ModifyDBSubnetGroup	ModifyEventSubscription
ModifyOptionGroup	PromoteReadReplica
PurchaseReservedDBInstancesOffering	RebootDBInstance
RemoveSourceIdentifierFromSubscription	RemoveTagsFromResource
ResetDBParameterGroup	RestoreDBInstanceFromDBSnapshot
RestoreDBInstanceToPointInTime	RevokeDBSecurityGroupIngress

Amazon Redshift

This guide focuses on the AWS SDK for PHP client for Amazon Redshift. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Redshift\RedshiftClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `eu-west-1`

```
use Aws\Redshift\RedshiftClient;

$client = RedshiftClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Redshift is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
```

```
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Redshift');
```

Creating a cluster

The primary resource in Amazon Redshift is the cluster. To create a cluster you will use the `CreateCluster` operation. There are several parameters you can send when creating a cluster, so please refer to the API docs to determine which parameters to use. The following is basic example.

```
$client->createCluster(array(
    'ClusterIdentifier'  => 'your-unique-cluster-id',
    'ClusterType'        => 'multi-node',
    'MasterUsername'     => 'yourusername',
    'MasterUserPassword' => 'Y0urP@$$w0rd',
    'NodeType'           => 'dw.hs1.xlarge',
    'NumberOfNodes'      => 2,
));
```

After the `CreateCluster` operation is complete, the record for your cluster will exist, but it will still take some time before the cluster is actually available for use. You can describe your cluster to check it's status.

```
$result = $client->describeClusters(array(
    'ClusterIdentifier' => 'your-unique-cluster-id',
));
$clusters = $result->get('Clusters');
$status = $clusters['ClusterStatus'];
```

If you would like your code to wait until the cluster is available, you can use the the `ClusterAvailable` waiter.

```
$client->waitUntilClusterAvailable(array(
    'ClusterIdentifier' => 'your-unique-cluster-id',
));
```

Warning

It can take over 20 minutes for a cluster to become available.

Creating snapshots

You can also take snapshots of your cluster with the `CreateClusterSnapshot` operation. Snapshots can take a little time before they become available as well, so there is a corresponding `SnapshotAvailable` waiter.

```
$client->createClusterSnapshot(array(
    'ClusterIdentifier'  => 'your-unique-cluster-id',
    'SnapshotIdentifier' => 'your-unique-snapshot-id',
));
$client->waitUntilSnapshotAvailable(array(
    'SnapshotIdentifier' => 'your-unique-snapshot-id',
));
```

Events

Amazon Redshift records events that take place with your clusters and account. These events are available for up to 14 days and can be retrieved via the `DescribeEvents` operation. Only 100 events can be returned at a time, so using the SDK's iterators feature allows you to easily fetch and iterate over all the events in your query without having to manually send repeated requests. The `StartTime` and `EndTime` parameters can take any PHP date string or DateTime object.

```
$events = $client->getIterator('DescribeEvents', array(
    'StartTime'  => strtotime('-3 days'),
    'EndTime'    => strtotime('now'),
));

foreach ($events as $event) {
    echo "{$event['Date']}: {$event['Message']}\n";
}
```

API Reference

Please see the Amazon Redshift Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AuthorizeClusterSecurityGroupIngress	AuthorizeSnapshotAccess
CopyClusterSnapshot	CreateCluster
CreateClusterParameterGroup	CreateClusterSecurityGroup
CreateClusterSnapshot	CreateClusterSubnetGroup
CreateEventSubscription	CreateHsmClientCertificate
CreateHsmConfiguration	DeleteCluster
DeleteClusterParameterGroup	DeleteClusterSecurityGroup
DeleteClusterSnapshot	DeleteClusterSubnetGroup
DeleteEventSubscription	DeleteHsmClientCertificate
DeleteHsmConfiguration	DescribeClusterParameterGroups
DescribeClusterParameters	DescribeClusterSecurityGroups
DescribeClusterSnapshots	DescribeClusterSubnetGroups
DescribeClusterVersions	DescribeClusters
DescribeDefaultClusterParameters	DescribeEventCategories
DescribeEventSubscriptions	DescribeEvents
DescribeHsmClientCertificates	DescribeHsmConfigurations
DescribeLoggingStatus	DescribeOrderableClusterOptions
DescribeReservedNodeOfferings	DescribeReservedNodes
DescribeResize	DisableLogging
DisableSnapshotCopy	EnableLogging
EnableSnapshotCopy	ModifyCluster
ModifyClusterParameterGroup	ModifyClusterSubnetGroup
ModifyEventSubscription	ModifySnapshotCopyRetentionPeriod
PurchaseReservedNodeOffering	RebootCluster
ResetClusterParameterGroup	RestoreFromClusterSnapshot
RevokeClusterSecurityGroupIngress	RevokeSnapshotAccess
RotateEncryptionKey	

Amazon Route 53

This guide focuses on the AWS SDK for PHP client for Amazon Route 53. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Route53\Route53Client::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\Route53\Route53Client;

$client = Route53Client::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Route 53 is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Route53');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Route 53 Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

ChangeResourceRecordSets	CreateHealthCheck
CreateHostedZone	DeleteHealthCheck
DeleteHostedZone	GetChange
GetHealthCheck	GetHostedZone
ListHealthChecks	ListHostedZones
ListResourceRecordSets	

Amazon Simple Storage Service

This guide focuses on the AWS SDK for PHP client for Amazon Simple Storage Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\S3\S3Client::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\S3\S3Client;

$client = S3Client::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Simple Storage Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('S3');
```

Creating a bucket

Now that we've created a client object, let's create a bucket. This bucket will be used throughout the remainder of this guide.

```
$client->createBucket(array('Bucket' => 'mybucket'));
```

If you run the above code example unaltered, you'll probably trigger the following exception:

```
PHP Fatal error:  Uncaught Aws\S3\Exception\BucketAlreadyExistsException: AWS Error
Code: BucketAlreadyExists, Status Code: 409, AWS Request ID: D94E6394791E98A4,
AWS Error Type: client, AWS Error Message: The requested bucket name is not
available. The bucket namespace is shared by all users of the system. Please select
a different name and try again.
```

This is because bucket names in Amazon S3 reside in a global namespace. You'll need to change the actual name of the bucket used in the examples of this tutorial in order for them to work correctly.

Creating a bucket in another region

The above example creates a bucket in the standard US-EAST-1 region. You can change the bucket location by passing a `LocationConstraint` value.

```
// Create a valid bucket and use a LocationConstraint
$result = $client->createBucket(array(
    'Bucket'             => $bucket,
    'LocationConstraint' => \Aws\Common\Enum\Region::US_WEST_2
));

// Get the Location header of the response
echo $result['Location'] . "\n";

// Get the request ID
echo $result['RequestId'] . "\n";
```

You'll notice in the above example that we are using the `Aws\Common\Enum\Region` object to provide the `US_WEST_2` constant. The SDK provides various Enum classes under the `Aws\Common\Enum` namespace that can be useful for remembering available values and ensuring you do not enter a typo.

Note

Using the enum classes is not required. You could just pass 'us-west-2' in the `LocationConstraint` key.

Waiting until the bucket exists

Now that we've created a bucket, let's force our application to wait until the bucket exists. This can be done easily using a *waiter*. The following snippet of code will poll the bucket until it exists or the maximum number of polling attempts are completed.

```
// Poll the bucket until it is accessible
$client->waitUntilBucketExists(array('Bucket' => $bucket));
```

Uploading objects

Now that you've created a bucket, let's put some data in it. The following example creates an object in your bucket called data.txt that contains 'Hello!'.

```
// Upload an object to Amazon S3
$result = $client->putObject(array(
    'Bucket' => $bucket,
    'Key'    => 'data.txt',
    'Body'   => 'Hello!'
));

// Access parts of the result object
echo $result['Expiration'] . "\n";
echo $result['ServerSideEncryption'] . "\n";
echo $result['ETag'] . "\n";
echo $result['VersionId'] . "\n";
echo $result['RequestId'] . "\n";

// Get the URL the object can be downloaded from
echo $result['ObjectURL'] . "\n";
```

The AWS SDK for PHP will attempt to automatically determine the most appropriate Content-Type header used to store the object. If you are using a less common file extension and your Content-Type header is not added automatically, you can add a Content-Type header by passing a `ContentType` option to the operation.

Uploading a file

The above example uploaded text data to your object. You can alternatively upload the contents of a file by passing the `SourceFile` option. Let's also put some metadata on the object.

```
// Upload an object by streaming the contents of a file
// $pathToFile should be absolute path to a file on disk
$result = $client->putObject(array(
    'Bucket'     => $bucket,
    'Key'        => 'data_from_file.txt',
    'SourceFile' => $pathToFile,
    'Metadata'   => array(
        'Foo' => 'abc',
        'Baz' => '123'
    )
));

// We can poll the object until it is accessible
$client->waitUntilObjectExists(array(
    'Bucket' => $this->bucket,
    'Key'    => 'data_from_file.txt'
));
```

Uploading from a stream

Alternatively, you can pass a resource returned from an `fopen` call to the `Body` parameter.

```
// Upload an object by streaming the contents of a PHP stream.
// Note: You must supply a "ContentLength" parameter to an
// operation if the steam does not respond to fstat() or if the
// fstat() of stream does not provide a valid the 'size' attribute.
// For example, the "http" stream wrapper will require a ContentLength
// parameter because it does not respond to fstat().
$client->putObject(array(
    'Bucket' => $bucket,
    'Key'    => 'data_from_stream.txt',
    'Body'   => fopen($pathToFile, 'r+')
));
```

Because the AWS SDK for PHP is built around Guzzle, you can also pass an EntityBody object.

```
// Be sure to add a use statement at the beginning of you script:
// use Guzzle\Http\EntityBody;

// Upload an object by streaming the contents of an EntityBody object
$client->putObject(array(
    'Bucket' => $bucket,
    'Key'    => 'data_from_entity_body.txt',
    'Body'   => EntityBody::factory(fopen($pathToFile, 'r+'))
));
```

Listing your buckets

You can list all of the buckets owned by your account using the `listBuckets` method.

```
$result = $client->listBuckets();

foreach ($result['Buckets'] as $bucket) {
    // Each Bucket value will contain a Name and CreationDate
    echo "{$bucket['Name']} - {$bucket['CreationDate']}\n";
}
```

All service operation calls using the AWS SDK for PHP return a `Guzzle\Service\Resource\Model` object. This object contains all of the data returned from the service in a normalized array like object. The object also contains a `get()` method used to retrieve values from the model by name, and a `getPath()` method that can be used to retrieve nested values.

```
// Grab the nested Owner/ID value from the result model using getPath()
$result = $client->listBuckets();
echo $result->getPath('Owner/ID') . "\n";
```

Listing objects in your buckets

Listing objects is a lot easier in the new SDK thanks to *iterators*. You can list all of the objects in a bucket using the `ListObjectsIterator`.

```
$iterator = $client->getIterator('ListObjects', array(
    'Bucket' => $bucket
));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

Iterators will handle sending any required subsequent requests when a response is truncated. The ListObjects iterator works with other parameters too.

```
$iterator = $client->getIterator('ListObjects', array(
    'Bucket' => $bucket,
    'Prefix' => 'foo'
));

foreach ($iterator as $object) {
    echo $object['Key'] . "\n";
}
```

You can convert any iterator to an array using the `toArray()` method of the iterator.

Note

Converting an iterator to an array will load the entire contents of the iterator into memory.

Downloading objects

You can use the `GetObject` operation to download an object.

```
// Get an object using the getObject operation
$result = $client->getObject(array(
    'Bucket' => $bucket,
    'Key'    => 'data.txt'
));

// The 'Body' value of the result is an EntityBody object
echo get_class($result['Body']) . "\n";
// > Guzzle\Http\EntityBody

// The 'Body' value can be cast to a string
echo $result['Body'] . "\n";
// > Hello!
```

The contents of the object are stored in the `Body` parameter of the model object. Other parameters are stored in model including `ContentType`, `ContentLength`, `VersionId`, `ETag`, etc...

The `Body` parameter stores a reference to a `Guzzle\Http\EntityBody` object. The SDK will store the data in a temporary PHP stream by default. This will work for most use-cases and will automatically protect your application from attempting to download extremely large files into memory.

The EntityBody object has other nice features that allow you to read data using streams.

```
// Seek to the beginning of the stream
$result['Body']->rewind();

// Read the body off of the underlying stream in chunks
while ($data = $result['Body']->read(1024)) {
    echo $data;
}

// Cast the body to a primitive string
// Warning: This loads the entire contents into memory!
$bodyAsString = (string) $result['Body'];
```

Saving objects to a file

You can save the contents of an object to a file by setting the SaveAs parameter.

```
$result = $client->getObject(array(
    'Bucket' => $bucket,
    'Key'    => 'data.txt',
    'SaveAs' => '/tmp/data.txt'
));

// Contains an EntityBody that wraps a file resource of /tmp/data.txt
echo $result['Body']->getUri() . "\n";
// > /tmp/data.txt
```

Uploading large files using multipart uploads

Amazon S3 allows you to uploads large files in pieces. The AWS SDK for PHP provides an abstraction layer that makes it easier to upload large files using multipart upload.

```
use Aws\Common\Enum\Size;
use Aws\Common\Exception\MultipartUploadException;
use Aws\S3\Model\MultipartUpload\UploadBuilder;

$uploader = UploadBuilder::newInstance()
    ->setClient($client)
    ->setSource('/path/to/large/file.mov')
    ->setBucket('mybucket')
    ->setKey('my-object-key')
    ->setOption('Metadata', array('Foo' => 'Bar'))
    ->setOption('CacheControl', 'max-age=3600')
    ->build();

// Perform the upload. Abort the upload if something goes wrong
try {
    $uploader->upload();
    echo "Upload complete.\n";
} catch (MultipartUploadException $e) {
    $uploader->abort();
    echo "Upload failed.\n";
}
```

You can attempt to upload parts in parallel by specifying the concurrency option on the UploadBuilder object. The following example will create a transfer object that will attempt to upload three parts in parallel until the entire object has been uploaded.

```
$uploader = UploadBuilder::newInstance()
    ->setClient($client)
    ->setSource('/path/to/large/file.mov')
    ->setBucket('mybucket')
```

```
    ->setKey('my-object-key')
    ->setConcurrency(3)
    ->build();
```

You can use the `Aws\S3\S3Client::upload()` method if you just want to upload files and not worry if they are too large to send in a single PutObject operation or require a multipart upload.

```
$client->upload('bucket', 'key', 'object body', 'public-read');
```

Setting ACLs and Access Control Policies

You can specify a canned ACL on an object when uploading:

```
$client->putObject(array(
    'Bucket'     => 'mybucket',
    'Key'        => 'data.txt',
    'SourceFile' => '/path/to/data.txt',
    'ACL'        => 'public-read'
));
```

You can use the `Aws\S3\Enum\CannedAcl` object to provide canned ACL constants:

```
use Aws\S3\Enum\CannedAcl;

$client->putObject(array(
    'Bucket'     => 'mybucket',
    'Key'        => 'data.txt',
    'SourceFile' => '/path/to/data.txt',
    'ACL'        => CannedAcl::PUBLIC_READ
));
```

You can specify more complex ACLs using the `ACP` parameter when sending PutObject, CopyObject, CreateBucket, CreateMultipartUpload, PutBucketAcl, PutObjectAcl, and other operations that accept a canned ACL. Using the `ACP` parameter allows you specify more granular access control policies using a `Aws\S3\Model\Acp` object. The easiest way to create an Acp object is through the `Aws\S3\Model\AcpBuilder`.

```
use Aws\S3\Enum\Permission;
use Aws\S3\Enum\Group;
use Aws\S3\Model\AcpBuilder;

$acp = AcpBuilder::newInstance()
    ->setOwner($myOwnerId)
    ->addGrantForEmail(Permission::READ, 'test@example.com')
    ->addGrantForUser(Permission::FULL_CONTROL, 'user-id')
    ->addGrantForGroup(Permission::READ, Group::AUTHENTICATED_USERS)
    ->build();

$client->putObject(array(
    'Bucket'     => 'mybucket',
    'Key'        => 'data.txt',
    'SourceFile' => '/path/to/data.txt',
    'ACP'        => $acp
));
```

Creating a pre-signed URL

You can authenticate certain types of requests by passing the required information as query-string parameters instead of using the Authorization HTTP header. This is useful for enabling direct third-party browser access to your private Amazon S3 data, without proxying the request. The idea is to construct a "pre-signed" request and encode it as a URL that an end-user's browser can retrieve. Additionally, you can limit a pre-signed request by specifying an expiration time.

The most common scenario is creating a pre-signed URL to GET an object. The easiest way to do this is to use the `getObjectUrl` method of the Amazon S3 client. This same method can also be used to get an unsigned URL of a public S3 object.

```
// Get a plain URL for an Amazon S3 object
$plainUrl = $client->getObjectUrl($bucket, 'data.txt');
// > https://my-bucket.s3.amazonaws.com/data.txt

// Get a pre-signed URL for an Amazon S3 object
$signedUrl = $client->getObjectUrl($bucket, 'data.txt', '+10 minutes');
// > https://my-bucket.s3.amazonaws.com/data.txt?AWSAccessKeyId=[...]&Expires=[...]&Signatur

// Create a vanilla Guzzle HTTP client for accessing the URLs
$http = new \Guzzle\Http\Client;

// Try to get the plain URL. This should result in a 403 since the object is private
try {
    $response = $http->get($plainUrl)->send();
} catch (\Guzzle\Http\Exception\ClientErrorResponseException $e) {
    $response = $e->getResponse();
}
echo $response->getStatusCode();
// > 403

// Get the contents of the object using the pre-signed URL
$response = $http->get($signedUrl)->send();
echo $response->getBody();
// > Hello!
```

You can also create pre-signed URLs for any Amazon S3 operation using the `getCommand` method for creating a Guzzle command object and then calling the `createPresignedUrl()` method on the command.

```
// Get a command object from the client and pass in any options
// available in the GetObject command (e.g. ResponseContentDisposition)
$command = $client->getCommand('GetObject', array(
    'Bucket' => $bucket,
    'Key' => 'data.txt',
    'ResponseContentDisposition' => 'attachment; filename="data.txt"'
));

// Create a signed URL from the command object that will last for
// 10 minutes from the current time
$signedUrl = $command->createPresignedUrl('+10 minutes');

echo file_get_contents($signedUrl);
// > Hello!
```

If you need more flexibility in creating your pre-signed URL, then you can create a pre-signed URL for a completely custom `Guzzle\Http\Message\RequestInterface` object. You can use the `get()`, `post()`, `head()`, `put()`, and `delete()` methods of a client object to easily create a Guzzle request object.

```
$key = 'data.txt';
$url = "{$bucket}/{$key}";

// get() returns a Guzzle\Http\Message\Request object
$request = $client->get($url);

// Create a signed URL from a completely custom HTTP request that
// will last for 10 minutes from the current time
$signedUrl = $client->createPresignedUrl($request, '+10 minutes');
```

```
echo file_get_contents($signedUrl);
// > Hello!
```

Amazon S3 stream wrapper

The Amazon S3 stream wrapper allows you to store and retrieve data from Amazon S3 using built-in PHP functions like `file_get_contents`, `fopen`, `copy`, `rename`, `unlink`, `mkdir`, `rmdir`, etc.

See *Amazon S3 Stream Wrapper*.

Syncing data with Amazon S3

Uploading a directory to a bucket

Uploading a local directory to an Amazon S3 bucket is rather simple:

```
$client->uploadDirectory('/local/directory', 'my-bucket');
```

The `uploadDirectory()` method of a client will compare the contents of the local directory to the contents in the Amazon S3 bucket and only transfer files that have changed. While iterating over the keys in the bucket and comparing against the names of local files using a customizable filename to key converter, the changed files are added to an in memory queue and uploaded concurrently. When the size of a file exceeds a customizable `multipart_upload_size` parameter, the uploader will automatically upload the file using a multipart upload.

Customizing the upload sync

The method signature of the *uploadDirectory()* method allows for the following arguments:

```
public function uploadDirectory($directory, $bucket, $keyPrefix = null, array $options = arr
```

By specifying `$keyPrefix`, you can cause the uploaded objects to be placed under a virtual folder in the Amazon S3 bucket. For example, if the `$bucket` name is `my-bucket` and the `$keyPrefix` is 'testing/', then your files will be uploaded to `my-bucket` under the `testing/` virtual folder: `https://my-bucket.s3.amazonaws.com/testing/filename.txt`

The `uploadDirectory()` method also accepts an optional associative array of `$options` that can be used to further control the transfer.

params	Array of parameters to use with each `PutObject` or `CreateMultipartUpload` operation performed during the transfer. For example, you can specify an `ACL` key to change the ACL of each uploaded object. See PutObject for a list of available options.
base_dir	Base directory to remove from each object key. By default, the `$directory` passed into the `uploadDirectory()` method will be removed from each object key.
force	Set to true to upload every file, even if the file is already in Amazon S3 and has not changed.
concurrency	Maximum number of parallel uploads (defaults to 5)
debug	Set to true to enable debug mode to print information about each upload. Setting this value to an `fopen` resource will write the debug output to a stream rather than to `STDOUT`.

In the following example, a local directory is uploaded with each object stored in the bucket using a `public-read` ACL, 20 requests are sent in parallel, and debug information is printed to standard output as each request is transferred.

```
$dir = '/local/directory';
$bucket = 'my-bucket';
$keyPrefix = '';

$client->uploadDirectory($dir, $bucket, $keyPrefix, array(
    'params'      => array('ACL' => 'public-read'),
    'concurrency' => 20,
```

```
    'debug'       => true
));
```

More control with the UploadSyncBuilder

The `uploadDirectory()` method is an abstraction layer over the much more powerful `Aws\S3\Sync\UploadSyncBuilder`. You can use an `UploadSyncBuilder` object directly if you need more control over the transfer. Using an `UploadSyncBuilder` allows for the following advanced features:

- Can upload only files that match a glob expression
- Can upload only files that match a regular expression
- Can specify a custom `\Iterator` object to use to yield files to an `UploadSync` object. This can be used, for example, to filter out which files are transferred even further using something like the Symfony 2 Finder component.
- Can specify the `Aws\S3\Sync\FilenameConverterInterface` objects used to convert Amazon S3 object names to local filenames and vice versa. This can be useful if you require files to be renamed in a specific way.

```
use Aws\S3\Sync\UploadSyncBuilder;

UploadSyncBuilder::getInstance()
    ->setClient($client)
    ->setBucket('my-bucket')
    ->setAcl('public-read')
    ->uploadFromGlob('/path/to/file/*.php')
    ->build()
    ->transfer();
```

Downloading a bucket to a directory

You can download the objects stored in an Amazon S3 bucket using features similar to the `uploadDirectory()` method and the `UploadSyncBuilder`. You can download the entire contents of a bucket using the `Aws\S3\S3Client::downloadBucket()` method.

The following example will download all of the objects from `my-bucket` and store them in `/local/directory`. Object keys that are under virtual subfolders are converted into a nested directory structure when downloading the objects. Any directories missing on the local filesystem will be created automatically.

```
$client->downloadBucket('/local/directory', 'my-bucket');
```

Customizing the download sync

The method signature of the `downloadBucket()` method allows for the following arguments:

```
public function downloadBucket($directory, $bucket, $keyPrefix = null, array $options = arra
```

By specifying `$keyPrefix`, you can limit the downloaded objects to only keys that begin with the specified `$keyPrefix`. This, for example, can be useful for downloading objects under a specific virtual directory.

The `downloadBucket()` method also accepts an optional associative array of `$options` that can be used to further control the transfer.

params	Array of parameters to use with each `GetObject` operation performed during the transfer. See GetObject for a list of available options.
base_dir	Base directory to remove from each object key when downloading. By default, the entire object key is used to determine the path to the file on the local filesystem.
force	Set to true to download every file, even if the file is already on the local filesystem and has not changed.
concurrency	Maximum number of parallel downloads (defaults to 10)

debug	Set to true to enable debug mode to print information about each download. Setting this value to an `fopen` resource will write the debug output to a stream rather than to `STDOUT`.
allow_resumable	Set to true to allow previously interrupted downloads to be resumed using a Range GET

More control with the DownloadSyncBuilder

The `downloadBucket()` method is an abstraction layer over the much more powerful `Aws\S3\Sync\DownloadSyncBuilder`. You can use a `DownloadSyncBuilder` object directly if you need more control over the transfer. Using the `DownloadSyncBuilder` allows for the following advanced features:

- Can download only files that match a regular expression
- Just like the `UploadSyncBuilder`, you can specify a custom `\Iterator` object to use to yield files to a `DownloadSync` object.
- Can specify the `Aws\S3\Sync\FilenameConverterInterface` objects used to convert Amazon S3 object names to local filenames and vice versa.

```
use Aws\S3\Sync\DownloadSyncBuilder;

DownloadSyncBuilder::getInstance()
    ->setClient($client)
    ->setDirectory('/path/to/directory')
    ->setBucket('my-bucket')
    ->setKeyPrefix('/under-prefix')
    ->allowResumableDownloads()
    ->build()
    ->transfer();
```

Cleaning up

Now that we've taken a tour of how you can use the Amazon S3 client, let's clean up any resources we may have created.

```
// Delete the objects in the bucket before attempting to delete
// the bucket
$clear = new ClearBucket($client, $bucket);
$clear->clear();

// Delete the bucket
$client->deleteBucket(array('Bucket' => $bucket));

// Wait until the bucket is not accessible
$client->waitUntilBucketNotExists(array('Bucket' => $bucket));
```

API Reference

Please see the Amazon Simple Storage Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AbortMultipartUpload (service docs)	CompleteMultipartUpload (service docs)
CopyObject (service docs)	CreateBucket (service docs)
CreateMultipartUpload (service docs)	DeleteBucket (service docs)
DeleteBucketCors (service docs)	DeleteBucketLifecycle (service docs)
DeleteBucketPolicy (service docs)	DeleteBucketTagging (service docs)
DeleteBucketWebsite (service docs)	DeleteObject (service docs)
DeleteObjects (service docs)	GetBucketAcl (service docs)
GetBucketCors (service docs)	GetBucketLifecycle (service docs)

GetBucketLocation (service docs)	GetBucketLogging (service docs)
GetBucketNotification (service docs)	GetBucketPolicy (service docs)
GetBucketRequestPayment (service docs)	GetBucketTagging (service docs)
GetBucketVersioning (service docs)	GetBucketWebsite (service docs)
GetObject (service docs)	GetObjectAcl (service docs)
GetObjectTorrent (service docs)	HeadBucket (service docs)
HeadObject (service docs)	ListBuckets (service docs)
ListMultipartUploads (service docs)	ListObjectVersions (service docs)
ListObjects (service docs)	ListParts (service docs)
PutBucketAcl (service docs)	PutBucketCors (service docs)
PutBucketLifecycle (service docs)	PutBucketLogging (service docs)
PutBucketNotification (service docs)	PutBucketPolicy (service docs)
PutBucketRequestPayment (service docs)	PutBucketTagging (service docs)
PutBucketVersioning (service docs)	PutBucketWebsite (service docs)
PutObject (service docs)	PutObjectAcl (service docs)
RestoreObject (service docs)	UploadPart (service docs)
UploadPartCopy (service docs)	

Amazon Simple Email Service

This guide focuses on the AWS SDK for PHP client for Amazon Simple Email Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Ses\SesClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-west-2`, `us-east-1`, `eu-west-1`

```
use Aws\Ses\SesClient;

$client = SesClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Simple Email Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Ses');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Simple Email Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

DeleteIdentity	DeleteVerifiedEmailAddress
GetIdentityDkimAttributes	GetIdentityNotificationAttributes
GetIdentityVerificationAttributes	GetSendQuota
GetSendStatistics	ListIdentities
ListVerifiedEmailAddresses	SendEmail
SendRawEmail	SetIdentityDkimEnabled
SetIdentityFeedbackForwardingEnabled	SetIdentityNotificationTopic
VerifyDomainDkim	VerifyDomainIdentity
VerifyEmailAddress	VerifyEmailIdentity

Amazon SimpleDB

This guide focuses on the AWS SDK for PHP client for Amazon SimpleDB. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\SimpleDb\SimpleDbClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `eu-west-1`

```
use Aws\SimpleDb\SimpleDbClient;

$client = SimpleDbClient::factory(array(
```

```
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon SimpleDB is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('SimpleDb');
```

Creating domains

The first step in storing data within Amazon SimpleDB is to create one or more domains.

Domains are similar to database tables, except that you cannot perform functions across multiple domains, such as querying multiple domains or using foreign keys. As a consequence, you should plan an Amazon SimpleDB data architecture that will meet the needs of your project.

Let's use the CreateDomain operation of the Amazon SimpleDB client to create a domain.

```
$client->createDomain(array('DomainName' => 'mydomain'));
```

List all domains

Now that the domain is created, we can list the domains in our account to verify that it exists. This is done using the ListDomains operation and the ListDomains iterator.

```
$domains = $client->getIterator('ListDomains')->toArray();
var_export($domains);
// Lists an array of domain names, including "mydomain"
```

Retrieving a domain

You can get more information about a domain using the DomainMetadata operation. This operation returns information about a domain, including when the domain was created, the number of items and attributes, and the size of attribute names and values.

```
$result = $client->domainMetadata(array('DomainName' => 'mydomain'));
echo $result['ItemCount'] . "\n";
echo $result['ItemNamesSizeBytes'] . "\n";
echo $result['AttributeNameCount'] . "\n";
echo $result['AttributeNamesSizeBytes'] . "\n";
echo $result['AttributeValueCount'] . "\n";
echo $result['AttributeValuesSizeBytes'] . "\n";
echo $result['Timestamp'] . "\n";
```

Adding items

After creating a domain, you are ready to start putting data into it. Domains consist of items, which are described by attribute name-value pairs. Items are added to a domain using the PutAttributes operation.

```
$client->putAttributes(array(
    'DomainName' => 'mydomain',
    'ItemName'   => 'test',
    'Attributes' => array(
        array('Name' => 'a', 'Value' => 1, 'Replace' => true),
        array('Name' => 'b', 'Value' => 2),
    )
));
```

Note

When you put attributes, notice that the Replace parameter is optional, and set to false by default. If you do not explicitly set Replace to true, a new attribute name-value pair is created each time; even if the Name value already exists in your Amazon SimpleDB domain.

Retrieving items

GetAttributes

We can check to see if the item was added correctly by retrieving the specific item by name using the GetAttribute operation.

```
$result = $client->getAttributes(array(
    'DomainName' => 'mydomain',
    'ItemName'   => 'test',
    'Attributes' => array(
        'a', 'b'
    ),
    'ConsistentRead' => true
));
```

Notice that we set the *ConsistentRead* option to *true*. Amazon SimpleDB keeps multiple copies of each domain. A successful write (using PutAttributes, BatchPutAttributes, DeleteAttributes, BatchDeleteAttributes, CreateDomain, or DeleteDomain) guarantees that all copies of the domain will durably persist. Amazon SimpleDB supports two read consistency options: eventually consistent read and consistent read. A consistent read (using Select or GetAttributes with ConsistentRead=true) returns a result that reflects all writes that received a successful response prior to the read.

You can find out more about consistency and Amazon SimpleDB in the service's developer guide on consistency.

Select

You can retrieve attributes for items by name, but Amazon SimpleDB also supports the Select operation. The Select operation returns a set of Attributes for ItemNames that match the select expression. Select is similar to the standard SQL SELECT statement.

Let's write a select query that will return all items withe the *a* attribute set to *1*.

```
$result = $client->select(array(
    'SelectExpression' => "select * from mydomain where a = '1'"
));
foreach ($result['Items'] as $item) {
    echo $item['Name'] . "\n";
    var_export($item['Attributes']);
}
```

Because some responses will be truncated and require subsequent requests, it is recommended to always use the Select iterator to easily retrieve an entire result set.

```
$iterator = $client->getIterator('Select', array(
    'SelectExpression' => "select * from mydomain where a = '1'"
));
foreach ($iterator as $item) {
    echo $item['Name'] . "\n";
    var_export($item['Attributes']);
}
```

You can find much more information about the Select operation in the service's developer guide on select.

Deleting items

You can delete specific attributes of an item or an entire item using the DeleteAttributes operation. If all attributes of an item are deleted, the item is deleted.

Let's go ahead and delete the item we created in *mydomain*.

```
$client->deleteAttributes(array(
    'DomainName' => 'mydomain',
    'ItemName'   => 'test'
));
```

Because we did not specify an *Attributes* parameter, the entire item is deleted.

Deleting domains

Now that we've explored some of the features of Amazon SimpleDB, we should delete our testing data. The DeleteDomain operation deletes a domain. Any items (and their attributes) in the domain are deleted as well. The DeleteDomain operation might take 10 or more seconds to complete.

```
$client->deleteDomain(array('DomainName' => 'mydomain'));
```

API Reference

Please see the Amazon SimpleDB Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

BatchDeleteAttributes	BatchPutAttributes
CreateDomain	DeleteAttributes
DeleteDomain	DomainMetadata
GetAttributes	ListDomains
PutAttributes	Select

Amazon Simple Notification Service

This guide focuses on the AWS SDK for PHP client for Amazon Simple Notification Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Sns\SnsClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Sns\SnsClient;

$client = SnsClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Simple Notification Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Sns');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Simple Notification Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddPermission	ConfirmSubscription
CreatePlatformApplication	CreatePlatformEndpoint
CreateTopic	DeleteEndpoint
DeletePlatformApplication	DeleteTopic
GetEndpointAttributes	GetPlatformApplicationAttributes
GetSubscriptionAttributes	GetTopicAttributes
ListEndpointsByPlatformApplication	ListPlatformApplications
ListSubscriptions	ListSubscriptionsByTopic
ListTopics	Publish

RemovePermission	SetEndpointAttributes
SetPlatformApplicationAttributes	SetSubscriptionAttributes
SetTopicAttributes	Subscribe
Unsubscribe	

Amazon Simple Queue Service

This guide focuses on the AWS SDK for PHP client for Amazon Simple Queue Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Sqs\SqsClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Sqs\SqsClient;

$client = SqsClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Simple Queue Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Sqs');
```

Creating a queue

Now, let's create a queue. You can create a standard queue by just providing a name. Make sure to get the queue's URL from the result, since the queue URL is the unique identifier used to specify the queue in order to send and receive messages.

```
$result = $client->createQueue(array('QueueName' => 'my-queue'));
$queueUrl = $result->get('QueueUrl');
```

You can also set attributes on your queue when you create it.

```
use Aws\Common\Enum\Size;
use Aws\Sqs\Enum\QueueAttribute;

$result = $client->createQueue(array(
    'QueueName'  => 'my-queue',
    'Attributes' => array(
        QueueAttribute::DELAY_SECONDS        => 5,
        QueueAttribute::MAXIMUM_MESSAGE_SIZE => 4 * Size::KB,
    ),
));
$queueUrl = $result->get('QueueUrl');
```

Or you can also set queue attributes later.

```
use Aws\Common\Enum\Time;
use Aws\Sqs\Enum\QueueAttribute;

$result = $client->setQueueAttributes(array(
    'QueueUrl'   => $queueUrl,
    'Attributes' => array(
        QueueAttribute::VISIBILITY_TIMEOUT => 2 * Time::MINUTES,
    ),
));
```

Sending messages

Sending a message to a queue is straight forward with the `SendMessage` command.

```
$client->sendMessage(array(
    'QueueUrl'    => $queueUrl,
    'MessageBody' => 'An awesome message!',
));
```

You can overwrite the queue's default delay for a message when you send it.

```
$client->sendMessage(array(
    'QueueUrl'     => $queueUrl,
    'MessageBody'  => 'An awesome message!',
    'DelaySeconds' => 30,
));
```

Receiving messages

Receiving messages is done with the `ReceiveMessage` command.

```
$result = $client->receiveMessage(array(
    'QueueUrl' => $queueUrl,
));

foreach ($result->getPath('Messages/*/Body') as $messageBody) {
    // Do something with the message
    echo $messageBody;
}
```

By default, only one message will be returned. If you want to get more messages, make sure to use the `MaxNumberOfMessages` parameter and specify a number of messages (1 to 10). Remember that you are not guaranteed to receive that many messages, but you can receive up to that amount depending on how many are actually in the queue at the time of your request.

SQS also supports "long polling", meaning that you can instruct SQS to hold the connection open with the SDK for up to 20 seconds in order to wait for a message to arrive in the queue. To configure this behavior, you must use the `WaitTimeSeconds` parameter.

```
$result = $client->receiveMessage(array(
    'QueueUrl'        => $queueUrl,
    'WaitTimeSeconds' => 10,
));
```

Note

You can also configure long-polling at the queue level by setting the `ReceiveMessageWaitTimeSeconds` queue attribute.

API Reference

Please see the Amazon Simple Queue Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddPermission	ChangeMessageVisibility
ChangeMessageVisibilityBatch	CreateQueue
DeleteMessage	DeleteMessageBatch
DeleteQueue	GetQueueAttributes
GetQueueUrl	ListDeadLetterSourceQueues
ListQueues	ReceiveMessage
RemovePermission	SendMessage
SendMessageBatch	SetQueueAttributes

AWS Storage Gateway

This guide focuses on the AWS SDK for PHP client for AWS Storage Gateway. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\StorageGateway\StorageGatewayClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\StorageGateway\StorageGatewayClient;

$client = StorageGatewayClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Storage Gateway is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('StorageGateway');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Storage Gateway Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

ActivateGateway	AddCache
AddUploadBuffer	AddWorkingStorage
CancelArchival	CancelRetrieval
CreateCachediSCSIVolume	CreateSnapshot
CreateSnapshotFromVolumeRecoveryPoint	CreateStorediSCSIVolume
CreateTapes	DeleteBandwidthRateLimit
DeleteChapCredentials	DeleteGateway
DeleteSnapshotSchedule	DeleteTape
DeleteTapeArchive	DeleteVolume
DescribeBandwidthRateLimit	DescribeCache
DescribeCachediSCSIVolumes	DescribeChapCredentials
DescribeGatewayInformation	DescribeMaintenanceStartTime
DescribeSnapshotSchedule	DescribeStorediSCSIVolumes
DescribeTapeArchives	DescribeTapeRecoveryPoints
DescribeTapes	DescribeUploadBuffer
DescribeVTLDevices	DescribeWorkingStorage
DisableGateway	ListGateways
ListLocalDisks	ListVolumeRecoveryPoints
ListVolumes	RetrieveTapeArchive

RetrieveTapeRecoveryPoint	ShutdownGateway
StartGateway	UpdateBandwidthRateLimit
UpdateChapCredentials	UpdateGatewayInformation
UpdateGatewaySoftwareNow	UpdateMaintenanceStartTime
UpdateSnapshotSchedule	

AWS Security Token Service

This guide focuses on the AWS SDK for PHP client for AWS Security Token Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Sts\StsClient::factory()` method and provide your credentials (`key` and `secret`).

```
use Aws\Sts\StsClient;

$client = StsClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Security Token Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Sts');
```

Note

For information about why you might need to use temporary credentials in your application or project, see Scenarios for Granting Temporary Access in the AWS STS documentation.

Getting Temporary Credentials

AWS STS has five operations that return temporary credentials: `AssumeRole`, `AssumeRoleWithWebIdentity`, `AssumeRoleWithSAML`, `GetFederationToken`, and `GetSessionToken`. Using the `GetSessionToken` operation is trivial, so let's use that one as an example.

```
$result = $client->getSessionToken();
```

The result for `GetSessionToken` and the other AWS STS operations always contains a `'Credentials'` value. If you print the result (e.g., `print_r($result)`), it looks like the following:

```
Array
(
    ...
    [Credentials] => Array
    (
        [SessionToken] => '<base64 encoded session token value>'
        [SecretAccessKey] => '<temporary secret access key value>'
        [Expiration] => 2013-11-01T01:57:52Z
        [AccessKeyId] => '<temporary access key value>'
    )
    ...
)
```

Using Temporary Credentials

You can use temporary credentials with another AWS client by instantiating the client and passing in the values received from AWS STS directly.

```
use Aws\S3\S3Client;

$result = $client->getSessionToken();

$s3 = S3Client::factory(array(
    'key'    => $result['Credentials']['AccessKeyId'],
    'secret' => $result['Credentials']['SecretAccessKey'],
    'token'  => $result['Credentials']['SessionToken'],
));
```

You can also construct a `Credentials` object and use that when instantiating the client.

```
use Aws\Common\Credentials\Credentials;
use Aws\S3\S3Client;

$result = $client->getSessionToken();

$credentials = new Credentials(
    $result['Credentials']['AccessKeyId'],
    $result['Credentials']['SecretAccessKey'],
    $result['Credentials']['SessionToken']
);

$s3 = S3Client::factory(array('credentials' => $credentials));
```

However, the *best* way to provide temporary credentials is to use the `createCredentials()` helper method included with `StsClient`. This method extracts the data from an AWS STS result and creates the `Credentials` object for you.

```
$result = $sts->getSessionToken();
$credentials = $sts->createCredentials($result);

$s3 = S3Client::factory(array('credentials' => $credentials));
```

You can also use the same technique when setting credentials on an existing client object.

```
$credentials = $sts->createCredentials($sts->getSessionToken());
$s3->setCredentials($credentials);
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Security Token Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AssumeRole	AssumeRoleWithSAML
AssumeRoleWithWebIdentity	DecodeAuthorizationMessage
GetFederationToken	GetSessionToken

AWS Support

This guide focuses on the AWS SDK for PHP client for AWS Support. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Support\SupportClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`

```
use Aws\Support\SupportClient;

$client = SupportClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to AWS Support is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Support');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the AWS Support Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

AddCommunicationToCase	CreateCase
DescribeCases	DescribeCommunications
DescribeServices	DescribeSeverityLevels
DescribeTrustedAdvisorCheckRefreshStatuses	DescribeTrustedAdvisorCheckResult
DescribeTrustedAdvisorCheckSummaries	DescribeTrustedAdvisorChecks
RefreshTrustedAdvisorCheck	ResolveCase

Amazon Simple Workflow Service

This guide focuses on the AWS SDK for PHP client for Amazon Simple Workflow Service. This guide assumes that you have already downloaded and installed the AWS SDK for PHP. See *Installation* for more information on getting started.

Creating a client

First you need to create a client object using one of the following techniques.

Factory method

The easiest way to get up and running quickly is to use the `Aws\Swf\SwfClient::factory()` method and provide your credentials (`key` and `secret`).

A `region` parameter is also required and must be set to one of the following values: `us-east-1`, `ap-northeast-1`, `sa-east-1`, `ap-southeast-1`, `ap-southeast-2`, `us-west-2`, `us-gov-west-1`, `us-west-1`, `cn-north-1`, `eu-west-1`

```
use Aws\Swf\SwfClient;

$client = SwfClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
```

```
    'region' => '<region name>'
));
```

You can provide your access keys like in the preceding example, or you can choose to omit them if you are using AWS Identity and Access Management (IAM) roles for EC2 instances or credentials sourced from the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables.

Service builder

A more robust way to connect to Amazon Simple Workflow Service is through the service builder. This allows you to specify credentials and other configuration settings in a configuration file. These settings can then be shared across all clients so that you only have to specify your settings once.

```
use Aws\Common\Aws;

// Create a service builder using a configuration file
$aws = Aws::factory('/path/to/my_config.json');

// Get the client from the builder by namespace
$client = $aws->get('Swf');
```

This guide is incomplete

This guide is not quite finished. If you are looking for a good way to contribute to the SDK and to the rest of the AWS PHP community, then helping to write documentation is a great place to start. Our guides are written in ReStructuredText and generated using Sphinx. Feel free to add some content to our documentation and send a pull request to https://github.com/aws/aws-sdk-php. You can view our documentation sources at https://github.com/aws/aws-sdk-php/tree/master/docs.

API Reference

Please see the Amazon Simple Workflow Service Client API reference for a details about all of the available methods, including descriptions of the inputs and outputs.

CountClosedWorkflowExecutions	CountOpenWorkflowExecutions
CountPendingActivityTasks	CountPendingDecisionTasks
DeprecateActivityType	DeprecateDomain
DeprecateWorkflowType	DescribeActivityType
DescribeDomain	DescribeWorkflowExecution
DescribeWorkflowType	GetWorkflowExecutionHistory
ListActivityTypes	ListClosedWorkflowExecutions
ListDomains	ListOpenWorkflowExecutions
ListWorkflowTypes	PollForActivityTask
PollForDecisionTask	RecordActivityTaskHeartbeat
RegisterActivityType	RegisterDomain
RegisterWorkflowType	RequestCancelWorkflowExecution
RespondActivityTaskCanceled	RespondActivityTaskCompleted
RespondActivityTaskFailed	RespondDecisionTaskCompleted

SignalWorkflowExecution	StartWorkflowExecution
TerminateWorkflowExecution	

DynamoDB Session Handler

Introduction

The **DynamoDB Session Handler** is a custom session handler for PHP that allows developers to use Amazon DynamoDB as a session store. Using DynamoDB for session storage alleviates issues that occur with session handling in a distributed web application by moving sessions off of the local file system and into a shared location. DynamoDB is fast, scalable, easy to setup, and handles replication of your data automatically.

The DynamoDB Session Handler uses the `session_set_save_handler()` function to hook DynamoDB operations into PHP's native session functions to allow for a true drop in replacement. This includes support for features like session locking and garbage collection which are a part of PHP's default session handler.

For more information on the Amazon DynamoDB service, please visit the Amazon DynamoDB homepage.

Basic Usage

1. Register the handler

The first step is to instantiate the Amazon DynamoDB client and register the session handler.

```
require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;

$dynamoDb = DynamoDbClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>'
));

$sessionHandler = $dynamoDb->registerSessionHandler(array(
    'table_name' => 'sessions'
));
```

You can also instantiate the `SessionHandler` object directly using it's `factory` method.

```
require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;
use Aws\DynamoDb\Session\SessionHandler;

$dynamoDb = DynamoDbClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>',
));

$sessionHandler = SessionHandler::factory(array(
    'dynamodb_client' => $dynamoDb,
    'table_name'      => 'sessions',
));
$sessionHandler->register();
```

2. Create a table for storing your sessions

Before you can actually use the session handler, you need to create a table in which to store the sessions. This can be done ahead of time through the AWS Console for Amazon DynamoDB, or you can use the session handler object (which you've already configured with the table name) by doing the following:

```
$sessionHandler->createSessionsTable(5, 5);
```

The two parameters for this function are used to specify the read and write provisioned throughput for the table, respectively.

> ***Note***
>
> The `createSessionsTable` function uses the `TableExists` *waiter* internally, so this function call will block until the table exists and is ready to be used.

3. Use PHP sessions like normal

Once the session handler is registered and the table exists, you can write to and read from the session using the `$_SESSION` superglobal, just like you normally do with PHP's default session handler. The DynamoDB Session Handler encapsulates and abstracts the interactions with Amazon DynamoDB and enables you to simply use PHP's native session functions and interface.

```
// Start the session
session_start();

// Alter the session data
$_SESSION['user.name'] = 'jeremy';
$_SESSION['user.role'] = 'admin';

// Close the session (optional, but recommended)
session_write_close();
```

Configuration

You may configure the behavior of the session handler using the following options. All options are optional, but you should make sure to understand what the defaults are.

`table_name`	The name of the DynamoDB table in which to store the sessions. This defaults to `sessions`.
`hash_key`	The name of the hash key in the DynamoDB sessions table. This defaults to `id`.
`session_lifetime`	The lifetime of an inactive session before it should be garbage collected. If it is not provided, then the actual lifetime value that will be used is `ini_get('session.gc_maxlifetime')`.
`consistent_read`	Whether or not the session handler should use consistent reads for the `GetItem` operation. This defaults to `true`.
`locking_strategy`	The strategy used for doing session locking. By default the handler uses the `NullLockingStrategy`, which means that session locking is **not** enabled (see the *Session Locking* section for more information). Valid values for this option include null, 'null', 'pessemistic', or an instance of `NullLockingStrategy` or `PessimisticLockingStrategy`.
`automatic_gc`	Whether or not to use PHP's session auto garbage collection. This defaults to the value of `(bool) ini_get('session.gc_probability')`, but the recommended value is `false`. (see the *Garbage Collection* section for more information).

`gc_batch_size`	The batch size used for removing expired sessions during garbage collection. This defaults to `25`, which is the maximum size of a single `BatchWriteItem` operation. This value should also take your provisioned throughput into account as well as the timing of your garbage collection.
`gc_operation_delay`	The delay (in seconds) between service operations performed during garbage collection. This defaults to `0`. Increasing this value allows you to throttle your own requests in an attempt to stay within your provisioned throughput capacity during garbage collection.
`max_lock_wait_time`	Maximum time (in seconds) that the session handler should wait to acquire a lock before giving up. This defaults to `10` and is only used with the `PessimisticLockingStrategy`.
`min_lock_retry_microtime`	Minimum time (in microseconds) that the session handler should wait between attempts to acquire a lock. This defaults to `10000` and is only used with the `PessimisticLockingStrategy`.
`max_lock_retry_microtime`	Maximum time (in microseconds) that the session handler should wait between attempts to acquire a lock. This defaults to `50000` and is only used with the `PessimisticLockingStrategy`.
`dynamodb_client`	The `DynamoDbClient` object that should be used for performing DynamoDB operations. If you register the session handler from a client object using the `registerSessionHandler()` method, this will default to the client you are registering it from. If using the `SessionHandler::factory()` method, you are required to provide an instance of `DynamoDbClient`.

To configure the Session Handler, you must specify the configuration options when you instantiate the handler. The following code is an example with all of the configuration options specified.

```
$sessionHandler = $dynamoDb->registerSessionHandler(array(
    'table_name'               => 'sessions',
    'hash_key'                 => 'id',
    'session_lifetime'         => 3600,
    'consistent_read'          => true,
    'locking_strategy'         => null,
    'automatic_gc'             => 0,
    'gc_batch_size'            => 50,
    'max_lock_wait_time'       => 15,
    'min_lock_retry_microtime' => 5000,
    'max_lock_retry_microtime' => 50000,
));
```

Pricing

Aside from data storage and data transfer fees, the costs associated with using Amazon DynamoDB are calculated based on the provisioned throughput capacity of your table (see the Amazon DynamoDB pricing details). Throughput is measured in units of Write Capacity and Read Capacity. The Amazon DynamoDB homepage says:

> A unit of Write Capacity enables you to perform one write per second for items of up to 1KB in size. Similarly, a unit of Read Capacity enables you to perform one strongly consistent read per second (or two eventually consistent reads per second) of items of up to 1KB in size. Larger items will require more capacity. You can calculate the number of units of read and write capacity you need by estimating the number of reads or writes you need to do per second and multiplying by the size of your items (rounded up to the nearest KB).

Ultimately, the throughput and the costs required for your sessions table is going to correlate with your expected traffic and session size. The following table explains the amount of read and write operations that are performed on your DynamoDB table for each of the session functions.

Read via `session_start()` (Using `NullLockingStrategy`)	• 1 read operation (only 0.5 if `consistent_read` is `false`). • (Conditional) 1 write operation to delete the session if it is expired.

Read via `session_start()` (Using `PessimisticLockingStrategy`)	• A minimum of 1 *write* operation. • (Conditional) Additional write operations for each attempt at acquiring a lock on the session. Based on configured lock wait time and retry options. • (Conditional) 1 write operation to delete the session if it is expired.
Write via `session_write_close()`	• 1 write operation.
Delete via `session_destroy()`	• 1 write operation.
Garbage Collection	• 0.5 read operations **per KB of data in the table** to scan for expired sessions. • 1 write operation **per expired item** to delete it.

Session Locking

The DynamoDB Session Handler supports pessimistic session locking in order to mimic the behavior of PHP's default session handler. By default the DynamoDB Session Handler has this feature *turned off* since it can become a performance bottleneck and drive up costs, especially when an application accesses the session when using ajax requests or iframes. You should carefully consider whether or not your application requires session locking or not before enabling it.

By default the session handler uses the `NullLockingStrategy` which does not do any session locking. To enable session locking, you should use the `PessimisticLockingStrategy`, which can be specified when the session handler is created.

```
$sessionHandler = $dynamoDb->registerSessionHandler(array(
    'table_name'       => 'sessions',
    'locking_strategy' => 'pessimistic',
));
```

Garbage Collection

The DynamoDB Session Handler supports session garbage collection by using a series of `Scan` and `BatchWriteItem` operations. Due to the nature of how the `Scan` operation works and in order to find all of the expired sessions and delete them, the garbage collection process can require a lot of provisioned throughput.

For this reason it is discouraged to rely on the PHP's normal session garbage collection triggers (i.e., the `session.gc_probability` and `session.gc_divisor` ini settings). A better practice is to set `session.gc_probability` to `0` and schedule the garbage collection to occur during an off-peak time where a burst of consumed throughput will not disrupt the rest of the application.

For example, you could have a nightly cron job trigger a script to run the garbage collection. This script might look something like the following:

```
require 'vendor/autoload.php';

use Aws\DynamoDb\DynamoDbClient;
use Aws\DynamoDb\Session\SessionHandler;

$dynamoDb = DynamoDbClient::factory(array(
    'key'    => '<aws access key>',
    'secret' => '<aws secret key>',
    'region' => '<region name>',
));

$sessionHandler = SessionHandler::factory(array(
    'dynamodb_client' => $dynamoDb,
    'table_name'      => 'sessions',
```

```
));

$sessionHandler->garbageCollect();
```

You can also use the `gc_operation_delay` configuration option on the session handler to introduce delays in between the `Scan` and `BatchWriteItem` operations that are performed by the garbage collection process. This will increase the amount of time it takes the garbage collection to complete, but it can help you spread out the requests made by the session handler in order to help you stay close to or within your provisioned throughput capacity during garbage collection.

Best Practices

1. Create your sessions table in a region that is geographically closest to or in the same region as your application servers. This will ensure the lowest latency between your application and DynamoDB database.
2. Choose the provisioned throughput capacity of your sessions table carefully, taking into account the expected traffic to your application and the expected size of your sessions.
3. Monitor your consumed throughput through the AWS Management Console or with Amazon CloudWatch and adjust your throughput settings as needed to meet the demands of your application.
4. Keep the size of your sessions small. Sessions that are less than 1KB will perform better and require less provisioned throughput capacity.
5. Do not use session locking unless your application requires it.
6. Instead of using PHP's built-in session garbage collection triggers, schedule your garbage collection via a cron job, or another scheduling mechanism, to run during off-peak hours. Use the `gc_operation_delay` option to add delays in between the requests performed for the garbage collection process.

Amazon S3 Stream Wrapper

Introduction

The Amazon S3 stream wrapper allows you to store and retrieve data from Amazon S3 using built-in PHP functions like `file_get_contents`, `fopen`, `copy`, `rename`, `unlink`, `mkdir`, `rmdir`, etc.

You need to register the Amazon S3 stream wrapper in order to use it:

```
// Register the stream wrapper from an S3Client object
$client->registerStreamWrapper();
```

This allows you to access buckets and objects stored in Amazon S3 using the `s3://` protocol. The "s3" stream wrapper accepts strings that contain a bucket name followed by a forward slash and an optional object key or prefix: `s3://<bucket>[/<key-or-prefix>]`.

Downloading data

You can grab the contents of an object using `file_get_contents`. Be careful with this function though; it loads the entire contents of the object into memory.

```
// Download the body of the "key" object in the "bucket" bucket
$data = file_get_contents('s3://bucket/key');
```

Use `fopen()` when working with larger files or if you need to stream data from Amazon S3.

```
// Open a stream in read-only mode
if ($stream = fopen('s3://bucket/key', 'r')) {
    // While the stream is still open
    while (!feof($stream)) {
        // Read 1024 bytes from the stream
        echo fread($stream, 1024);
```

```
    }
    // Be sure to close the stream resource when you're done with it
    fclose($stream);
}
```

Opening Seekable streams

Streams opened in "r" mode only allow data to be read from the stream, and are not seekable by default. This is so that data can be downloaded from Amazon S3 in a truly streaming manner where previously read bytes do not need to be buffered into memory. If you need a stream to be seekable, you can pass `seekable` into the stream context options of a function.

```
$context = stream_context_create(array(
    's3' => array(
        'seekable' => true
    )
));

if ($stream = fopen('s3://bucket/key', 'r', false, $context)) {
    // Read bytes from the stream
    fread($stream, 1024);
    // Seek back to the beginning of the stream
    fseek($steam, 0);
    // Read the same bytes that were previously read
    fread($stream, 1024);
    fclose($stream);
}
```

Opening seekable streams allows you to seek only to bytes that were previously read. You cannot skip ahead to bytes that have not yet been read from the remote server. In order to allow previously read data to recalled, data is buffered in a PHP temp stream using Guzzle's CachingEntityBody decorator. When the amount of cached data exceed 2MB, the data in the temp stream will transfer from memory to disk. Keep this in mind when downloading large files from Amazon S3 using the `seekable` stream context setting.

Uploading data

Data can be uploaded to Amazon S3 using `file_put_contents()`.

```
file_put_contents('s3://bucket/key', 'Hello!');
```

You can upload larger files by streaming data using `fopen()` and a "w", "x", or "a" stream access mode. The Amazon S3 stream wrapper does **not** support simultaneous read and write streams (e.g. "r+", "w+", etc). This is because the HTTP protocol does not allow simultaneous reading and writing.

```
$stream = fopen('s3://bucket/key', 'w');
fwrite($stream, 'Hello!');
fclose($stream);
```

Note

Because Amazon S3 requires a Content-Length header to be specified before the payload of a request is sent, the data to be uploaded in a PutObject operation is internally buffered using a PHP temp stream until the stream is flushed or closed.

fopen modes

PHP's fopen() function requires that a `$mode` option is specified. The mode option specifies whether or not data can be read or written to a stream and if the file must exist when opening a stream. The Amazon S3 stream wrapper supports the following modes:

r	A read only stream where the file must already exist.
w	A write only stream. If the file already exists it will be overwritten.
a	A write only stream. If the file already exists, it will be downloaded to a temporary stream and any writes to the stream will be appended to any previously uploaded data.
x	A write only stream. An error is raised if the file does not already exist.

Other object functions

Stream wrappers allow many different built-in PHP functions to work with a custom system like Amazon S3. Here are some of the functions that the Amazon S3 stream wrapper allows you to perform with objects stored in Amazon S3.

unlink()	Delete an object from a bucket.
	`// Delete an object from a bucket`
	`unlink('s3://bucket/key');`
	
	You can pass in any options available to the `DeleteObject` operation to modify how the object is deleted (e.g. specifying a specific object version).
	`// Delete a specific version of an object from a bucket`
	`unlink('s3://bucket/key', stream_context_create(array(`
	`    's3' => array('VersionId' => '123')`
	`));`

filesize()	Get the size of an object.
	`// Get the Content-Length of an object`
	`$size = filesize('s3://bucket/key', );`

is_file()	Checks if a URL is a file.
	`if (is_file('s3://bucket/key')) {`
	`    echo 'It is a file!';`
	`}`

file_exists()	Checks if an object exists.
	`if (file_exists('s3://bucket/key')) {`
	`    echo 'It exists!';`
	`}`

filetype()	Checks if a URL maps to a file or bucket (dir).
file()	Load the contents of an object in an array of lines. You can pass in any options available to the `GetObject` operation to modify how the file is downloaded.
filemtime()	Get the last modified date of an object.
rename()	Rename an object by copying the object then deleting the original. You can pass in options available to the `CopyObject` and `DeleteObject` operations to the stream context parameters to modify how the object is copied and deleted.

copy()	Copy an object from one location to another. You can pass options available to the `CopyObject` operation into the stream context options to modify how the object is copied.
	`// Copy a file on Amazon S3 to another bucket`
	`copy('s3://bucket/key', 's3://other_bucket/key');`

Working with buckets

You can modify and browse Amazon S3 buckets similar to how PHP allows the modification and traversal of directories on your filesystem.

Here's an example of creating a bucket:

```
mkdir('s3://bucket');
```

You can pass in stream context options to the `mkdir()` method to modify how the bucket is created using the parameters available to the CreateBucket operation.

```
// Create a bucket in the EU region
mkdir('s3://bucket', stream_context_create(array(
    's3' => array(
        'LocationConstraint' => 'eu-west-1'
    )
));
```

You can delete buckets using the `rmdir()` function.

```
// Delete a bucket
rmdir('s3://bucket');
```

Note

A bucket can only be deleted if it is empty.

Listing the contents of a bucket

The opendir(), readdir(), rewinddir(), and closedir() PHP functions can be used with the Amazon S3 stream wrapper to traverse the contents of a bucket. You can pass in parameters available to the ListObjects operation as custom stream context options to the `opendir()` function to modify how objects are listed.

```
$dir = "s3://bucket/";

if (is_dir($dir) && ($dh = opendir($dir))) {
    while (($file = readdir($dh)) !== false) {
        echo "filename: {$file} : filetype: " . filetype($dir . $file) . "\n";
    }
    closedir($dh);
}
```

You can recursively list each object and prefix in a bucket using PHP's RecursiveDirectoryIterator.

```
$dir = 's3://bucket';
$iterator = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($dir));

foreach ($iterator as $file) {
    echo $file->getType() . ': ' . $file . "\n";
}
```

Another easy way to list the contents of the bucket is using the Symfony2 Finder component.

```php
<?php

require 'vendor/autoload.php';

use Symfony\Component\Finder\Finder;

$aws = Aws\Common\Aws::factory('/path/to/config.json');
$s3 = $aws->get('s3')->registerStreamWrapper();

$finder = new Finder();

// Get all files and folders (key prefixes) from "bucket" that are less than 100k
// and have been updated in the last year
$finder->in('s3://bucket')
    ->size('< 100K')
    ->date('since 1 year ago');

foreach ($finder as $file) {
    echo $file->getType() . ": {$file}\n";
}
```

The **AWS SDK for PHP** enables PHP developers to use Amazon Web Services from their PHP code, and build robust applications and software using services like Amazon S3, Amazon DynamoDB, Amazon Glacier, etc. You can get started in minutes by installing the SDK through Composer — by requiring the `aws/aws-sdk-php` package — or by downloading the standalone aws.zip or aws.phar files.

Getting Started

- Before you use the SDK
  - Sign up for AWS and get your AWS access keys
  - *Verify that your system meets the minimum requirements for the SDK*
  - *Install the AWS SDK for PHP*
- Using the SDK
  - *Getting Started Guide* – Everything you need to know to use the AWS SDK for PHP
  - Sample Project
- Migrating from Version 1 of the SDK?
  - *Migration Guide* – Migrating from Version 1 of the SDK to Version 2
  - *Side-by-side Guide* – Using Version 1 and Version 2 of the SDK side-by-side in the same project

In-Depth Guides

- *Providing Credentials to the SDK*
- *Configuring the SDK*

- SDK Features
  - *Iterators*
  - *Waiters*
  - *Command Objects*
  - *Parallel Commands*
  - *Modeled Responses*
  - *Static Client Facades*
- *Frequently Asked Questions (FAQ)*
- *Performance Guide*
- Contributing to the SDK
- Guzzle Documentation

Service-Specific Guides

- Amazon CloudFront
  - *Using the CloudFront PHP client*
  - PHP API reference
  - *Using the older 2012-05-05 API version*
- Amazon CloudSearch
  - *Using the Amazon CloudSearch PHP client*
  - PHP API reference
- Amazon CloudWatch
  - *Using the CloudWatch PHP client*
  - PHP API reference
- Amazon DynamoDB
  - *Using the DynamoDB PHP client*
  - PHP API reference
  - *Special Feature: DynamoDB Session Handler*
  - *Using the older 2011-12-05 API version*
- Amazon Elastic Compute Cloud (Amazon EC2)
  - *Using the Amazon EC2 PHP client*
  - PHP API reference
- Amazon Elastic MapReduce (Amazon EMR)
  - *Using the Amazon EMR PHP client*
  - PHP API reference
- Amazon Elastic Transcoder
  - *Using the Amazon Elastic Transcoder PHP client*
  - PHP API reference
- Amazon ElastiCache
  - *Using the Amazon ElastiCache PHP client*
  - PHP API reference

- Amazon Glacier
  - *Using the Amazon Glacier PHP client*
  - PHP API reference
- Amazon Kinesis
  - *Using the Kinesis PHP client*
  - PHP API reference
- Amazon Redshift
  - *Using the Amazon Redshift PHP client*
  - PHP API reference
- Amazon Relational Database Service (Amazon RDS)
  - *Using the Amazon RDS PHP client*
  - PHP API reference
- Amazon Route 53
  - *Using the Route 53 PHP client*
  - PHP API reference
- Amazon Simple Email Service (Amazon SES)
  - *Using the Amazon SES PHP client*
  - PHP API reference
- Amazon Simple Notification Service (Amazon SNS)
  - *Using the Amazon SNS PHP client*
  - PHP API reference
- Amazon Simple Queue Service (Amazon SQS)
  - *Using the Amazon SQS PHP client*
  - PHP API reference
- Amazon Simple Storage Service (Amazon S3)
  - *Using the Amazon S3 PHP client*
  - PHP API reference
  - *Special Feature: Amazon S3 Stream Wrapper*
- Amazon Simple Workflow Service (Amazon SWF)
  - *Using the Amazon SWF PHP client*
  - PHP API reference
- Amazon SimpleDB
  - *Using the Amazon SimpleDB PHP client*
  - PHP API reference
- Auto Scaling
  - *Using the Auto Scaling PHP client*
  - PHP API reference
- AWS CloudFormation
  - *Using the AWS CloudFormation PHP client*
  - PHP API reference
- AWS CloudTrail

    - *Using the CloudTrail PHP client*
    - PHP API reference
- AWS Data Pipeline
    - *Using the AWS Data Pipeline PHP client*
    - PHP API reference
- AWS Direct Connect
    - *Using the AWS Direct Connect PHP client*
    - PHP API reference
- AWS Elastic Beanstalk
    - *Using the Elastic Beanstalk PHP client*
    - PHP API reference
- AWS Identity and Access Management (AWS IAM)
    - *Using the IAM PHP client*
    - PHP API reference
- AWS Import/Export
    - *Using the AWS Import/Export PHP client*
    - PHP API reference
- AWS OpsWorks
    - *Using the AWS OpsWorks PHP client*
    - PHP API reference
- AWS Security Token Service (AWS STS)
    - *Using the AWS STS PHP client*
    - PHP API reference
- AWS Storage Gateway
    - *Using the AWS Storage Gateway PHP client*
    - PHP API reference
- AWS Support
    - *Using the AWS Support PHP client*
    - PHP API reference
- Elastic Load Balancing
    - *Using the Elastic Load Balancing PHP client*
    - PHP API reference

Articles from the Blog

- Syncing Data with Amazon S3
- Amazon S3 PHP Stream Wrapper
- Transferring Files To and From Amazon S3
- Provision an Amazon EC2 Instance with PHP
- Uploading Archives to Amazon Glacier from PHP
- Using AWS CloudTrail in PHP - Part 1
- Using AWS CloudTrail in PHP - Part 2

- Providing credentials to the AWS SDK for PHP
- Using Credentials from AWS Security Token Service
- Iterating through Amazon DynamoDB Results
- Sending requests through a proxy
- Wire Logging in the AWS SDK for PHP
- Streaming Amazon S3 Objects From a Web Server
- Static Service Client Facades

Presentations

Slides

- Mastering the AWS SDK for PHP
- Getting Good with the AWS SDK for PHP
- Using DynamoDB with the AWS SDK for PHP
- Controlling the AWS Cloud with PHP

Videos

- Mastering the AWS SDK for PHP (AWS re:Invent 2013)
- Using DynamoDB with the AWS SDK for PHP (AWS re:Invent 2012)